Impact of Discretization Techniques on Nonlinear Model Reduction and Analysis of the Structure of the POD Basis

Benjamin Unger

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Jeffrey Todd Borggaard, Chair
John Allen Burns
Alexander Elgart

27 September 2013
Blacksburg, Virginia

Keywords: Nonlinear Model Reduction, POD, DEIM, Burgers' equation, Optimal Control

Impact of Discretization Techniques on Nonlinear Model Reduction and Analysis of the Structure of the POD Basis

Benjamin Unger

(ABSTRACT)

In this thesis a numerical study of the one dimensional viscous Burgers' equation is conducted. The discretization techniques Finite Differences, Finite Element Method and Group Finite Elements are applied and their impact on model reduction techniques, namely Proper Orthogonal Decomposition (POD), Group POD and the Discrete Empirical Interpolation Method (DEIM), is studied. This study is facilitated by examination of several common ODE solvers. Embedded in this process, some results on the structure of the POD basis and an alternative algorithm to compute the POD subspace are presented. Various numerical studies are conducted to compare the different methods and the to study the interaction of the spatial discretization on the ROM through the basis functions. Moreover, the results are used to investigate the impact of Reduced Order Models (ROM) on Optimal Control Problems. To this end, the ROM is embedded in a Trust Region Framework and the convergence results of Arian et al. (2000) is extended to POD-DEIM. Based on the convergence theorem and the results of the numerical studies, the emphasis is on implementation strategies for numerical speedup.

Acknowledgments

First, I would like to express my sincere gratitude to my advisor and committee chairman Dr. Jeff Borggaard. He gave me not only helpful inspiration, but also the freedom to realize my own ideas. His encouragement, insights, guidance and advice made my pursuit of research deeper than ever expected before. I also owe thanks to Dr. John Burns and Dr. Alexander Elgart for serving on my committee and the supportive discussions.

Thanks to the Department of Mathematics of the Karlsruhe Institute of Technology (KIT) and the Department of Mathematics of Virginia Polytechnic Institute and State University (Virginia Tech), I was allowed to participate in a financially supported and very rewarding exchange program. I want to acknowledge the scholarship program of the German Academic Exchange Program (DAAD), which made this year abroad possible. Special thanks to Prof. Dr. Wolfgang Reichel for his continuous guidance and patience in the application process.

I would like to thank the graduate student community within the Department of Mathematics at Virginia Tech for their encouragement, helpful discussions and guidance in the field and their personal advice. Our numerous discussions contributed a great deal toward my thesis. Thanks to my friends in the United States and in Germany for making my time at Virginia Tech most enjoyable. But most of all I want to thank my family. Without them none of this would have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction and Motivation

The study of fluid flows is carried out by numerous researchers all over the world. One of the most important equations regarding flows is the Navier-Stokes equation, which is a nonlinear partial differential equation (PDE). Although this equation has been the subject of many investigations, we still lack general existence and uniqueness results. To this end, numerical studies are important for simulation, analysis and control of fluid flows. However, numerical simulations of complex nonlinear systems can be computationally expensive due to large spatial and temporal grids employed in the discretization process. Fine grids are crucial in order to obtain a desired accuracy. However, fine spatial grids result in large-scale dynamical systems that are expensive in terms of time and storage costs. Model reduction is a strategy to resolve this issue and a variety of techniques have been investigated over the past two decades. In particular, the *Proper Orthogonal Decomposition* (POD) is widely used in the computational fluid dynamics and optimal control community [61], [44], [5], [1], [11], [2], [56], [53]. Still, nonlinear POD models may be intractable due to the added costs of approximating the nonlinear terms. The *Discrete Empirical Interpolation Method* (DEIM) [17], [18], [19], [20], is one approach to speed up the computation of the nonlinearities.

This thesis consists of three parts. First, the impact of discretization techniques on reduced order models (ROM) is examined. During this study, a deeper analysis of the structure of the POD basis is given, which represents the second part of the thesis. The PDE used for this investigation is the semilinear PDE

$$\frac{\partial}{\partial t}y(t,x) + y(t,x)\frac{\partial}{\partial x}y(t,x) - \nu\frac{\partial^2}{\partial x^2}y(t,x) = f(t,x),$$

which is known as *Burgers' equation*. Here, $\nu > 0$ is a given parameter, f is a forcing term and y the unknown solution. This parabolic equation contains a convective term $y(t,x)\frac{\partial}{\partial x}y(t,x)$ and a dissipative term $\frac{\partial^2}{\partial x^2}y(t,x)$. If ν tends to zero, hyperbolic effects become dominant, while the steady state problem is of elliptic form [57]. Burgers' equation is simple enough to provide insights into more complex fluid models such as the Navier-Stokes equation and

still serves as model for combustion and related problems. In particular, Burgers' equation has been applied to the study of traffic flow [39], parametric transmission [49] and more. The availability of solutions (see Section 1.3) and its approximating character of the one-dimensional unsteady Navier-Stokes equation [47] has made Burgers' equation a first case study for numerical algorithms concerning fluid flows. Finally, the third component of this thesis applies to previous results in a study of the optimal control of Burgers' equation. The distributed control problem is numerically investigated using a steepest descent algorithm [14], [41]. To guarantee convergence with the ROM, the gradient-based optimization is embedded in a trust region framework. This was first introduced by Arian, Fahl and Sachs [5] and is known as Trust Region POD (TRPOD).

The thesis is organized as follows. After this introduction, the notation is explained and Burgers' equation is introduced. The discretization process is investigated in Chapter 2. Besides common *Finite Difference* (FD) and *Finite Element Methods* (FEM), *Group Finite Elements* (GFE) are considered. Several works [26], [57], [65], [52] have shown the effectivity of this interpolation extension of FEM. This is followed by a discussion of Reduced Order Models, namely POD and POD-DEIM in Chapter 3. Subsection 3.1.3 provides a detailed analysis of the structure of the POD basis. As in the discretization of the PDE, we examine a modification of POD by grouping terms, which is known as Group POD (GPOD), introduced by Dickinson [22] and successfully implemented by [43],[38]. Chapter 4 illustrates the optimal control problem. The TRPOD algorithm is extended to POD-DEIM and illustrated with several numerical studies. The thesis concludes with a summary and short outlook in Chapter 5.

1.2 Notation

We first set some notational conventions to limit subsequent interruptions. All matrices (except vectors) are printed in bold face. The identity matrix of dimension $k \in \mathbb{N}$ is denoted by $\mathbf{I}_k$. The transpose of a matrix (or a vector) is denoted with superscript $\top$. For vectors we denote the weighted inner product with $\langle \cdot, \cdot \rangle_{\mathbf{M}}$ with symmetric positive definite mass matrix $\mathbf{M}$. More precisely for $u, v \in \mathbb{R}^N$ and $\mathbf{M} \in \mathbb{R}^{N \times N}$ we have

$$\langle u, v \rangle_{\mathbf{M}} = u^\top \mathbf{M} v.$$

In contrast, the entrywise vector product is denoted with $\bullet$, that is

$$\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} \bullet \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix} = \begin{pmatrix} u_1 v_1 \\ u_2 v_2 \\ \vdots \\ u_N v_N \end{pmatrix}.$$

Partial derivatives are denoted with subscripts, for example

$$\frac{\partial}{\partial t} y(t, x) = y_t(t, x) \tag{1.1}$$

denotes the derivative for the first component. If a function is time-dependent, the associated variable is denoted with t. Moreover, the time-dependent variable is always the first variable as for instance in (1.1).

The numerical analysis and variational formulations require tools from functional analysis. For any integer $n \in \mathbb{N}$ define

$$C^n(\Omega) = \{f : \Omega \to \mathbb{R} \quad | \quad D^\alpha f \text{ exists and is continuous on } \Omega \text{ for any multi-index } |\alpha| \leq n\}$$

for the domain $\Omega \subseteq \mathbb{R}^d$. The Hilbert space of the square integrable functions over the domain Ω is given by

$$L^2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \quad \text{such that} \quad \int_\Omega |f(x)|^2 dx < \infty \right\}$$

with inner product

$$\langle f, g \rangle_{L^2(\Omega)} = \int_\Omega f(x)\overline{g(x)}dx$$

and induced norm

$$\|f\|_{L^2(\Omega)} = \sqrt{\int_\Omega |f(x)|^2 dx}.$$

The measure considered throughout this thesis is the Lebesgue measure. The vector space of piecewise smooth functions over Ω is denoted with $PWS(\Omega)$. The restriction of a function space X to functions (or classes of functions) with compact support ist denoted by X_0. Finally, the Kronecker delta is defined as

$$\delta_{ij} := \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

1.3 Burgers' Equation

In this thesis we examine the one-dimensional viscid Burgers' equation. It is proven to be a second order approximation of the one-dimensional unsteady Navier-Stokes equation when relaxing the incompressibility constraint [47]. This fact, in combination with the ability to compute analytical solutions, is the main reason that Burgers' equation is subject to numerous studies, for example [40] and [55]. In particular, many researchers try to verify their methods with Burgers' equation. Throughout this paper we study Burgers' equation in the form

$$y_t + yy_x - \nu y_{xx} = f + u \qquad \text{on } (0,T) \times (0,1), \tag{1.2a}$$
$$y(t,0) = y(t,1) = 0 \qquad \text{for a.e. } t \in (0,T), \tag{1.2b}$$
$$y(0,x) = y_0(x) \qquad \text{for all } x \in [0,1]. \tag{1.2c}$$

Here, $f : (0,T) \times (0,1) \to \mathbb{R}$ is a forcing term, $u : (0,T) \times (0,1) \to \mathbb{R}$ the control, $\nu > 0$ a ratio of convective to diffusive terms, which sometimes is referred to as the reciprocal of the Reynolds number, and $y_0 : [0,1] \to \mathbb{R}$ the initial condition. We call u the *control* or

input and y the *state*. At this point we give a short and incomplete overview of some results associated with Burgers' equation. We shall start with the Hopf-Cole transformation ([21], [35]). For this purpose, assume that the forcing term f and the control u are identically zero. We can rewrite (1.2a) in the so called *conservation form*

$$y_t + \frac{1}{2}\frac{\partial}{\partial x}\left(y^2\right) = \nu y_{xx}.$$

Substituting $y = \varphi_x$ and integrating over the spatial domain yields

$$\varphi_t + \frac{1}{2}\varphi_x^2 = \nu \varphi_{xx}.$$

Applying the transformation $\varphi = -2\nu \log w$ results in the heat equation $w_t = \nu w_{xx}$ with initial condition $w_0(x) := \exp\left(-\frac{1}{2\nu}\int_0^x y_0(\xi)d\xi\right)$. The heat equation is a well studied linear PDE and can be solved with the separation of variables technique for a finite domain or as the Cauchy problem on the real line. There also exists several analytic results for existence and uniqueness of weak and even classical solutions for certain boundary conditions. Examples are given by Byrnes, Gilliam and Shubov [16] as well as Ito and Yan [37].

If ν approaches zero, the viscid Burgers' equation turns into the *inviscid* Burgers' equation

$$y_t + y y_x = 0. \tag{1.3}$$

From standard PDE theory, we know by the method of characteristics that even for smooth initial conditions the PDE has no longer a classical solution. Therefore the notion of solution is extended to include discontinuous solutions that satisfy the weak form of the PDE, these are called weak solutions. In our discussion of the proposed methods we therefore use a shock as initial condition to reflect this possibility.

Chapter 2

Discretization

Discretization is usually the first step in order to find a numerical approximation. Two popular methods are the Finite Differences Method (FD) and the Finite Element Method (FEM). Fletcher [26] outlined that grouping terms in the conservative form of Burger's equation has some computational advantages over the regular FEM. This method is referred to as Group Finite Elements (GFE). The studies of Pugh [57], Smith [65] and Nguyen [52] with different boundary conditions confirmed these advantages. Hence, we employ GFE as third discretization technique. For given $N \in \mathbb{N}$, all three approaches provide a spatial discretization that transforms the nonlinear Burgers' equation

$$y_t + yy_x - \nu y_{xx} = f(t,x) \qquad \text{on } (0,T) \times [0,1], \tag{2.1a}$$

$$y(t,0) = y(t,1) = 0 \qquad \text{for a.e. } t \in (0,T), \tag{2.1b}$$

$$y(0,x) = y_0(x) \qquad \text{for all } x \in [0,1]. \tag{2.1c}$$

into a nonlinear system of ODEs of the form

$$\mathbf{M}\dot{\alpha}(t) = \mathbf{A}\alpha(t) + \mathcal{N}(\alpha(t)) + \mathbf{B}\overline{f}(t), \tag{2.2}$$

where $\mathbf{M} \in \mathbb{R}^{N\times N}$ denotes the mass matrix, $\mathbf{A} \in \mathbb{R}^{N\times N}$ the linear discretization matrix, which is also called stiffness matrix or system matrix, $\mathcal{N} : \mathbb{N} \to \mathbb{N}$ a nonlinear function and $\mathbf{B} \in \mathbb{R}^{N\times N}$ the discretization matrix for the forcing term. This system of ODEs can be solved with a common ODE solver. In this chapter we apply popular adaptive MATLAB® solvers (ODE45, ODE23S, ODE15S) [63] as well as an own implementation of the backward (implicit) Euler with a fixed time grid. We obtain a spatial grid by dividing the spatial domain $\Omega = [0,1]$ into $N+1$ equally spaced subintervals $[x_i, x_{i+1}]$ of length $h = \frac{1}{N+1}$, where $x_i := ih = \frac{i}{N+1}$ for $i = 0, 1, \ldots, N+1$. Let

$$\overline{f}(t) = \begin{pmatrix} f(t,x_1) \\ f(t,x_2) \\ \vdots \\ f(t,x_N) \end{pmatrix} \in \mathbb{R}^N$$

be the discretized time-dependent forcing vector. In Section 2.1 we present a FD for Burgers' equation while Section 2.2 outlines the idea of FEM. This is followed by the derivation of the GFE in Section 2.3 and the respective numerical results are given in Section 2.7.

2.1 Finite Differences (FD)

FD is based on a Taylor expansion of the solution and well suited for test problems with standard domains. Although modern approaches for Burgers' equation use the Hopf-Cole transformation mentioned in the introduction (for example [40],[55]) we apply the FD scheme directly on (2.1a). For this purpose assume that y is a solution of (2.1) and recall the Taylor expansions

$$y(t,x+h) = y(t,x) + y_x(t,x)h + \frac{1}{2}y_{xx}(t,x)h^2 + \frac{1}{3!}y_{xxx}(t,x)h^3 + \mathcal{O}(h^4) \quad \text{for } h \to 0, \tag{2.3}$$

$$y(t,x-h) = y(t,x) - y_x(t,x)h + \frac{1}{2}y_{xx}(t,x)h^2 - \frac{1}{3!}y_{xxx}(t,x)h^3 + \mathcal{O}(h^4) \quad \text{for } h \to 0. \tag{2.4}$$

Subtracting (2.4) from (2.3) yields

$$y_x(t,x) = \frac{1}{2h}\left(y(t,x+h) - y(t,x-h)\right) + \mathcal{O}(h^2) \qquad \text{for } h \to 0,$$

while adding both equations results in

$$y_{xx}(t,x) = \frac{1}{h^2}\left(y(t,x-h) - 2y(t,x) + y(t,x+h)\right) + \mathcal{O}(h^2) \qquad \text{for } h \to 0.$$

Using the spatial grid as introduced in the beginning of this chapter, we infer the approximations

$$y_x(t,x_i) \approx \frac{y(t,x_{i+1}) - y(t,x_{i-1})}{2h} \qquad \text{for } i = 1,\dots,N, \qquad \text{and} \tag{2.5}$$

$$y_{xx}(t,x_i) \approx \frac{y(t,x_{i-1}) - 2y(t,x_i) + y(t,x_{i+1})}{h^2} \qquad \text{for } i = 1,\dots,N. \tag{2.6}$$

Substituting (2.5) and (2.6) into (2.1a) yields the approximating system

$$y_t(t,x_i) \approx \frac{\nu}{h^2}\left(y(t,x_{i-1}) - 2y(t,x_i) + y(t,x_{i+1})\right) - \frac{1}{2h}y(t,x_i)\left(y(t,x_{i+1}) - y(t,x_{i-1})\right) + f(t,x_i)$$

for $i = 1,\dots,N$, while the boundary condition (2.1b) implies $y(t,x_0) = 0 = y(t,x_{N+1})$. Let $y_i(t) \approx y(t,x_i)$ be the numerical solution of (2.1) for $i = 0,1,\dots,N,N+1$. With the time-dependent vector $\alpha(t) = \begin{pmatrix} y_1(t) & \cdots & y_N(t) \end{pmatrix}^\top$ and the discretized forcing term $\overline{f}(t) = \begin{pmatrix} f(t,x_1) & \cdots & f(t,x_N) \end{pmatrix}^\top$ we can approximate (2.1) as

$$\mathbf{M}_{\text{FD}}\dot{\alpha}(t) = \mathbf{A}_{\text{FD}}\alpha(t) + \mathcal{N}_{\text{FD}}\left(\alpha(t)\right) + \mathbf{B}_{\text{FD}}\overline{f}(t),$$

which is of the desired form (2.2). Here, $\mathbf{M}_{\text{FD}} = \mathbf{B}_{\text{FD}} = \mathbf{I}_N \in \mathbb{R}^N$ are the identity while $\mathbf{A}_{\text{FD}} \in \mathbb{R}^N$ and $\mathcal{N}_{\text{FD}}(\alpha(t))$ are given by

$$\mathbf{A}_{\text{FD}} = \frac{\nu}{h^2}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{N\times N}$$

and

$$\mathcal{N}_{\mathrm{FD}}\left(\alpha(t)\right) = \frac{1}{2h}\begin{pmatrix} -y_1(t)y_2(t) \\ y_2(t)\left(y_1(t) - y_3(t)\right) \\ \vdots \\ y_{N-1}(t)\left(y_{N-2}(t) - y_N(t)\right) \\ y_N(t)y_{N-1}(t) \end{pmatrix} \in \mathbb{R}^N.$$

Note that the nonlinear term can be implemented efficiently in MATLAB® by a pointwise vector matrix-vector product

$$\mathcal{N}_{\mathrm{FD}}\left(\alpha(t)\right) = \alpha(t) \bullet \left[\frac{1}{2h}\begin{pmatrix} 0 & -1 & & & \\ 1 & & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & & -1 \\ & & & 1 & 0 \end{pmatrix}\alpha(t)\right].$$

2.2 Finite Element Method (FEM)

The Finite Element Method (FEM) is a very well founded numerical method to find approximate solutions for PDEs. The basis for FEM is to solve an ordinary differential equation (ODE) which one gets from a Galerkin projection on a finite dimensional subspace. More generally, FEM requires the following steps:

1. Find a weak formulation of the PDE.
2. Introduce a Finite Element Basis corresponding to the spatial discretization.
3. Apply the Galerkin projection.

Assume $y(t,x)$ is a solution of (2.1). Then we can multiply (2.1a) with a test function $\phi \in PWS_0([0,1])$ and integrate over the spatial domain to obtain

$$\int_0^1 \left[y_t(t,x) + y(t,x)y_x(t,x) - \nu y_{xx}(t,x)\right]\phi(x)dx = \int_0^1 f(t,x)\phi(x)dx.$$

Integration by parts (or Green's theorem for a higher dimensional version) of the convective term $\nu y_{xx}(t,x)\phi(x)$ yields

$$\int_0^1 \left[y_t(t,x) + y(t,x)y_x(t,x)\right]\phi(x)dx + \nu\int_0^1 y_x(t,x)\phi'(x)dx = \int_0^1 f(t,x)\phi(x)dx, \tag{2.7}$$

where the boundary parts drop due to the compact support of the test function. Conversely, if (2.7) holds for all $\phi \in C_0^1([0,1])$, the fundamental theorem of Calculus of Variations and the du Bois-Reymond lemma implies that y solves (2.1a) almost everywhere.

Definition 2.1 (Weak Form)**.** *Equation* (2.7) *is called* weak form *or* variational form *of* (2.1a)*. A solution of* (2.7) *is called a* weak solution *of* (2.1a)*.*

The weak formulation yields the idea to solve (2.7) for specific test functions, the so called Finite Element Basis. For this purpose we introduce piecewise linear basis functions $\phi_i : [0,1] \to \mathbb{R}$ given by

$$\phi_0(x) = \begin{cases} -\frac{1}{h}(x - x_1), & \text{for } x_0 \leq x \leq, x_1 \\ 0, & \text{otherwise,} \end{cases}$$

$$\phi_i(x) = \begin{cases} \frac{1}{h}(x - x_{i-1}), & \text{for } x_{i-1} \leq x \leq x_i, \\ -\frac{1}{h}(x - x_{i+1}), & \text{for } x_i \leq x \leq x_{i+1}, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, \ldots, N$$

$$\phi_{N+1}(x) = \begin{cases} \frac{1}{h}(x - x_N), & \text{for } x_N \leq x \leq, x_{N+1} \\ 0, & \text{otherwise.} \end{cases}$$

By definition $\phi_i(x_j) = \delta_{ij}$ for $i, j = 0, \ldots, N+1$ and is linear in each subinterval $[x_j, x_{j+1}]$. Note that this basis depends on the choice of N. An explicit example of the linear basis functions, also called *hat functions*, is presented in Figure 2.1. At this point it is worth

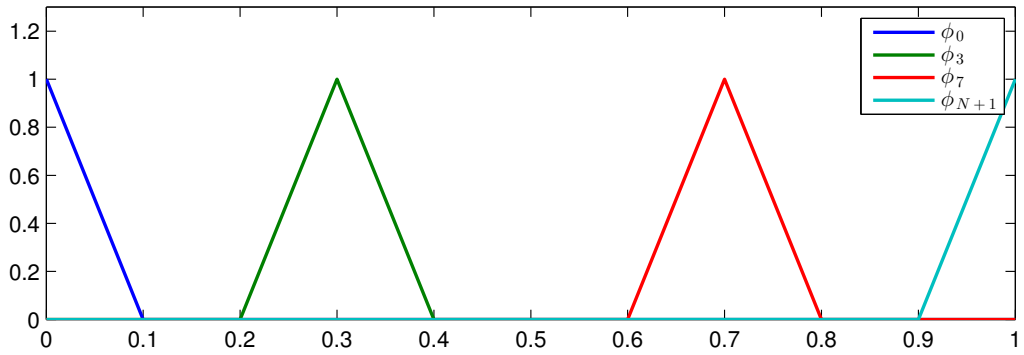


Figure 2.1: Linear basis functions, $N = 10$

mentioning that one can also use higher order basis functions (piecewise quadratic, cubic etc.), ormore smooth functions, such as Hermite cubics. The choice of linear basis functions instead of higher order finite elements is based on the studies mentioned in the beginning of this chapter. For a sufficiently good time discretization, the linear basis functions provide good numerical results so there is no need of additional smoothness.

Definition 2.2 (Finite Element Space)**.** *Let V^N denote the subspace spanned by the linear basis functions, i.e.*

$$V^N = span\{\phi_0, \phi_1, \ldots, \phi_{N+1}\}.$$

Then V^N is called a finite element space*.*

Even with this specific choice of test functions, the solution of the weak form (2.7) lives in an infinite dimensional function space. We apply a Galerkin projection to receive a problem that lives in a finite dimensional space. We project the solution of (2.1) on the $(N+2)$-dimensional finite element space V^N, i.e. we introduce the approximation

$$y(t,x) \approx y^N(t,x) = \mathcal{P}_{V^N} y(t,x) := \sum_{i=0}^{N+1} \alpha_i(t)\phi_i(x) \tag{2.8}$$

with unknown coefficients $\alpha_i : [0,T] \to \mathbb{R}$. Here, $\mathcal{P}_{Y^N}$ denotes the projection operator on the Finite Element space V^N. We substitute the approximation (2.8) into the weak form (2.7) along with the linear basis functions $\phi_i(x)$ to obtain

$$\int_0^1 \left[y_t^N(t,x) + y^N(t,x)y_x^N(t,x)\right]\phi_i(x)dx + \nu\int_0^1 y_x^N(t,x)\phi_i'(x)dx = \int_0^1 f(t,x)\phi_i(x)dx \tag{2.9}$$

for $i = 0,\dots,N+1$. Note that the boundary terms produced by integration by parts still vanish since either the basis functions are compactly supported or the solution is forced to be zero by the boundary conditions (2.1b). In particular, the homogeneous boundary conditions (2.1b) imply

$$0 = y^N(t,0) = \sum_{i=0}^{N+1} \alpha_i(t)\phi_i(0) = \alpha_0(t)$$

and

$$0 = y^N(t,1) = \sum_{i=0}^{N+1} \alpha_i(t)\phi_i(1) = \alpha_{N+1}(t).$$

Following Krämer [43] we give the following definition.

Definition 2.3 (Approximation of the weak solution)**.** *A function $y^N(t,\cdot) \in V^N$ is called* approximation of the weak solution *of* (2.1) *if $y^N(t,\cdot)$ satisfies* (2.9) *for $t > 0$, the boundary condition*

$$y^N(\cdot,0) = 0 = y^N(\cdot,1)$$

and the initial condition

$$y^N(0,\cdot) = P_{V^N} y_0(\cdot).$$

Recall that the approximation is given by $y^N(t,x) = \sum_{i=0}^{N=1} \alpha_i(t)\phi_i(x)$. Since $\alpha_0 \equiv 0 \equiv \alpha_{N+1}$ are already determined, we only have to find the unknown coefficient functions $\alpha_i(t)$ for $i = 1,\dots,N$. The derivatives of the approximation (2.8) are given by

$$y_t^N(t,x) = \sum_{i=1}^{N} \dot{\alpha}_i(t)\phi_i(x) \qquad \text{and} \qquad y_x^N(t,x) = \sum_{i=1}^{N} \alpha_i(t)\phi_i'(x).$$

The unknown coefficient functions $\alpha_i(t)$ $(i = 1,\dots,N)$ only depend on time, so the PDE (2.1) has been reduced to a system of ODEs. The final step is to rewrite (2.9) in the form of (2.2). It remains to compute the different parts of (2.9).

Mass Matrix We start with the term

$$\int_0^1 y_t^N(t,x)\phi_i(x)dx \qquad \text{for } i = 1,\dots,N.$$

In line with ODE theory we denote the resulting mass matrix with M_{FEM}. For fixed $i \in \{1,\dots,N\}$ we have

$$\int_0^1 y_t^N(t,x)\phi_i(x)dx = \int_0^1 \left(\sum_{j=1}^N \dot{\alpha}_j(t)\phi_j(x)\right)\phi_i(x)dx = \sum_{j=1}^N \dot{\alpha}_j(t)\underbrace{\int_0^1 \phi_j(x)\phi_i(x)dx}_{=:m_{ij}^{\text{FEM}}}. \tag{2.10}$$

Introducing the time-dependent vector $\alpha(t) = \begin{pmatrix}\alpha_1(t) & \cdots & \alpha_N(t)\end{pmatrix}^\top$ and the mass matrix $\mathbf{M}_{\text{FEM}} = \left(\left(m_{i,j}^{\text{FEM}}\right)\right)$ we can rewrite (2.10) as $\mathbf{M}_{\text{FEM}}\dot{\alpha}(t)$. As outlined in Appendix A, the mass matrix is given by

$$\mathbf{M}_{\text{FEM}} = \frac{1}{6(N+1)}\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Note that $\mathbf{M}_{\text{FEM}}$ is symmetric positive definite and regular. Hence, $\mathbf{M}_{\text{FEM}}$ has a Cholesky factorization.

Stiffness matrix The other linear term results in a so called *stiffness matrix*. A similar procedure as before yields

$$-\int_0^1 \nu y_x(t,x)\phi_i'(x)dx = \sum_{j=1}^N \alpha_j(t)\left(\underbrace{-\nu\int_0^1 \phi_i'(x)\phi_j'(x)dx}_{=:a_{ij}^{\text{FEM}}}\right)$$

with $\mathbf{A}_{\text{FEM}} = \left(\left(a_{ij}^{\text{FEM}}\right)\right)$ given by

$$\mathbf{A}_{\text{FEM}} = \nu(N+1)\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

The stiffness matrix depends on the parameter ν.

Nonlinear Term Let us now focus on the nonlinear term given by $-\int_0^1 y^N(t,x)y_x^N(t,x)\phi_i(x)$. Interchanging summation and integration yields

$$-\int_0^1 y^N(t,x)y_x^N(t,x)\phi_i(x) = -\sum_{j=1}^N\sum_{k=1}^N\left(\int_0^1 \phi_j(x)\phi_k'(x)\phi_i(x)dx\right)\alpha_j(t)\alpha_k(t)$$

for $i \in \{1,\dots,N\}$. Based on the calculations in Appendix A, the nonlinear term in the FEM approximation is given by

$$\mathcal{N}_{\text{FEM}}(\alpha(t)) = \frac{1}{6}\begin{pmatrix} -\alpha_1(t)\alpha_2(t) - \alpha_2^2(t) \\ \alpha_1^2(t) + \alpha_1(t)\alpha_2(t) - \alpha_2(t)\alpha_3(t) - \alpha_3^2(t) \\ \vdots \\ \alpha_{N-2}^2(t) + \alpha_{N-2}(t)\alpha_{N-1}(t) - \alpha_{N-1}(t)\alpha_N(t) - \alpha_N^2(t) \\ \alpha_{N-1}^2(t) + \alpha_{N-1}(t)\alpha_N(t) \end{pmatrix} \in \mathbb{R}^N.$$

By introducing the matrix $\mathbf{N}_{\text{FEM}} \in \mathbb{R}^{N\times N}$ given by

$$\mathbf{N}_{\text{FEM}} = \begin{pmatrix} 0 & -1 & & & \\ 1 & & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & & -1 \\ & & & 1 & 0 \end{pmatrix},$$

the nonlinear term can be implemented as

$$\mathcal{N}_{\text{FEM}}(\alpha(t)) = \frac{1}{6}\left(\mathbf{N}_{\text{FEM}}[\alpha(t)\bullet\alpha(t)] + \alpha(t)\bullet[\mathbf{N}_{\text{FEM}}\alpha(t)]\right). \tag{2.11}$$

Disregarding the prefactors $\frac{1}{2h}$ and $\frac{1}{6}$, the second summand equals the nonlinear term of the FD approximation.

Forcing Term We conclude the derivation of the system of ODEs by applying a quadrature formula to approximate the forcing term. Using the midpoint approximation

$$\int_a^b f(x)dx \approx (b-a)\cdot f\left(\frac{a+b}{2}\right) \tag{2.12}$$

and the definition of the linear basis functions we have

$$\begin{aligned}
\int_0^1 f(t,x)\phi_i(x)dx &= \int_{x_{i-1}}^{x_i} f(x)\phi_i(x)dx + \int_{x_i}^{x_{i+1}} f(x)\phi(x)dx \\
&\overset{(2.12)}{\approx} h\left(f\left(\frac{x_{i-1}+x_i}{2}\right)\underbrace{\phi_i\left(\frac{x_{i-1}+x_i}{2}\right)}_{\frac{1}{2}} + f\left(\frac{x_i+x_{i+1}}{2}\right)\underbrace{\phi_i\left(\frac{x_i+x_{i+i}}{2}\right)}_{\frac{1}{2}}\right) \\
&= \frac{h}{2}\left(f\left(\frac{x_{i-1}+x_i}{2}\right) + f\left(\frac{x_i+x_{i+1}}{2}\right)\right) \\
&\approx \frac{h}{4}\left(f(x_{i-1}) + 2f(x_i) + f(x_{i+1})\right).
\end{aligned}$$

The last approximation

$$f\left(\frac{a+b}{2}\right) \approx \frac{1}{2}\left(f(a)+f(b)\right)$$

is justified by the intermediate value theorem and a small step size h. Ignoring the boundary points and introducing the vector $\overline{f}(t) = \begin{pmatrix} f(t,x_1) & \cdots & f(t,x_N) \end{pmatrix}^\top \in \mathbb{R}^N$ we can approximate the forcing term by $\mathbf{B}_{\text{FEM}}\overline{f}(t)$, where

$$\mathbf{B}_{\text{FEM}} = \frac{1}{4(N+1)} \begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 2 & 1 \\ & & & 1 & 2 \end{pmatrix}.$$

Initial condition Finally, we have to take care of the initial condition (2.1c). Again, multiplying both sides with the basis functions and integrating over the spatial domain yields

$$\int_0^1 y(x,0)\phi_i(x)dx = \int_0^1 y_0(x)\phi_i(x)dx.$$

Hence, since the mass matrix $\mathbf{M}_{\text{FEM}}$ is invertible, we find

$$\alpha(0) = \begin{pmatrix} \alpha_1(0) \\ \vdots \\ \alpha_N(0) \end{pmatrix} = \mathbf{M}_{\text{FEM}}^{-1} \begin{pmatrix} \int_0^1 y_0(x)\phi_1(x)dx \\ \vdots \\ \int_0^1 y_0(x)\phi_N(x)dx \end{pmatrix} =: \overline{\alpha}_0$$

to be the projected initial condition. Note that one can either compute the initial condition analytically or use a quadrature formula analogously to the forcing term.

Putting all computations together, this finally implies that we have to solve the initial value problem

$$\begin{cases} \mathbf{M}_{\text{FEM}}\dot{\alpha}(t) = \mathbf{A}_{\text{FEM}}\alpha(t) + \mathcal{N}_{\text{FEM}}(\alpha(t)) + \mathbf{B}_{\text{FEM}}\overline{f}(t) & t \in [0,T], \\ \alpha(0) = \overline{\alpha}_0, \end{cases} \tag{2.13}$$

which coincides with the form of (2.2).

2.3 Group Finite Elements (GFE)

The studies of Fletcher [26], Pugh [57], Smith [65] and Nguyen [52] verified computational advantages over the regular FEM by using a group approximation of the nonlinearity. This method is referred to as Group Finite Elements (GFE). For its derivation we recall Burgers' equation in its conservative form

$$y_t + \frac{1}{2}\frac{\partial}{\partial x}\left(y^2\right) - \nu y_{xx} = f(t,x) \qquad \text{on } (0,T)\times[0,1], \tag{2.14a}$$

$$y(t,0) = y(t,1) = 0 \qquad \text{for a.e. } t \in (0,T), \tag{2.14b}$$

$$y(0,x) = y_0(x) \qquad \text{for all } x \in [0,1]. \tag{2.14c}$$

Similar to the previous section, the weak form is given by

$$\int_0^1 \left[y_t(t,x_n) + \frac{1}{2}\frac{d}{dx}\left(y(t,x)^2\right)\right]\phi(x)dx + \nu\int_0^1 y_x(t,x)\phi_x(x)dx = \int_0^1 f(t,x)\phi(x)dx. \quad (2.15)$$

We use the basis functions ϕ_i and the approximation (2.8) as before. Substitution into the weak form yields the same result as before in the linear terms. For the nonlinearity we observe the following. For $n = 1,\dots,N$ we have

$$y(t,x_n)^2 \approx [y^N(t,x)]^2 = \left[\sum_{i=1}^N \alpha_i(t)\phi_i(x_n)\right]^2 = \sum_{i=1}^N \alpha_i^2(t)\phi_i(x_n),$$

which implies that $\sum_{i=1}^N \alpha_i^2(t)\phi_i(x)$ is an interpolation of $\left[y^N(t,x)\right]^2$. Hence, it is reasonable to use the grouped approximation

$$y(t,x)^2 \approx \left[y^N(t,x)\right]^2 = \sum_{i=1}^N \alpha_i^2(t)\phi_i(x). \quad (2.16)$$

This results into a computational advantage in the nonlinear term, as we see in Section 2.7. Since only the nonlinear term changes compared to FEM, we have

$$\mathbf{M}_{\text{FEM}} = \mathbf{M}_{\text{GFE}}, \qquad \mathbf{A}_{\text{FEM}} = \mathbf{A}_{\text{GFE}} \qquad \text{and} \qquad \mathbf{B}_{\text{FEM}} = \mathbf{B}_{\text{GFE}}.$$

Nonlinear Term The nonlinear term is given by $\int_0^1 \frac{1}{2}\frac{\partial}{\partial x}\left([y^N(t,x)]^2\right)\phi_i(x)$. Applying the same process as before and using the grouped approximation $\left[y^N(t,x)\right]^2 \approx \sum_{j=1}^N \alpha_j^2(t)\phi_j(x)$ we compute

$$\int_0^1 \frac{1}{2}\frac{\partial}{\partial x}\left([y^N(t,x)]^2\right)\phi_i(x) = \sum_{j=1}^N \underbrace{\frac{1}{2}\left(\int_0^1 \phi_j'(x)\phi_i(x)dx\right)}_{=:n_{ij}^{\text{GFE}}}\alpha_j^2(t).$$

With the notation $\alpha(t)^2 := \alpha(t)\bullet\alpha(t)$, we can express the nonlinear term as

$$\mathcal{N}_{\text{GFE}}(\alpha(t)) = \mathbf{N}_{\text{GFE}}\alpha(t)^2,$$

where $\mathbf{N}_{\text{GFE}} \in \mathbb{R}^{N\times N}$ is given by (for detailed computations see Appendix A)

$$\mathbf{N}_{\text{GFE}} = \frac{1}{4}\begin{pmatrix} 0 & -1 & & & \\ 1 & 0 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & -1 \\ & & & 1 & 0 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

The matrix vector multiplication is easy to implement and the squared vector is computationally fast thanks to MATLAB® 's pointwise square implementation. Furthermore, we

recognize that ignoring the prefactors, this is again the same matrix as in the FD and FEM approximations of the nonlinearity. This time, the nonlinear term equals (beside some scaling factor) the first summand of the FEM approximation (2.11). It appears that the nonlinear term of the FEM approximation combines the FD and the GFE approach.

Finally, the initial-value problem with the GFE is given by

$$\begin{cases} \mathbf{M}_{\mathrm{GFE}}\dot{\alpha}(t) = \mathbf{A}_{\mathrm{GFE}}\alpha(t) + \mathbf{N}_{\mathrm{GFE}}\alpha^2(t) + \mathbf{B}_{\mathrm{GFE}}\overline{f}(t) & t \in [0,T], \\ \alpha(0) = \overline{\alpha}_0. \end{cases} \tag{2.17}$$

2.4 ODE Solver

Our aim is to solve system (2.2) with the derived discretization matrices and functions. We use MATLAB® to implement the algorithms. MATLAB® encloses some standard Runge-Kutta solvers, which we want to use in the first step. The solvers have an adaptive step size which improves the accuracy of the solution. Regarding the results of Jarvis [38] we use the ODE45 solver to obtain accurate approximations. Unfortunately, in the situation of optimal control, the control is only available on fixed time points. So we can not use an adaptive solver in this case and need to implement our own routine. For simplicity, we employ the backward Euler method, which is also known as implicit Euler. The reader is referred to Appendix B for implementation details.

We quickly review the idea of the implicit Euler. Consider the general initial value problem

$$\begin{cases} \dot{y}(t) = g(t, y(t)) \\ y(t_0) = y_0. \end{cases}$$

By the fundamental theorem of calculus it is

$$y(t) = y(t_0) + \int_{t_0}^{t} \dot{y}(s)ds = y(0) + \int_{t_0}^{t} g(s, y(s))ds.$$

For $M \in \mathbb{N}$, introduce the time grid $\{t_i\}_{i=1}^{M}$ with step size $h_i = t_i - t_{i-1}$ for $i = 2, \ldots, M$. We approximate the integral by

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} g(s, y(s))ds \approx y_0 + h_1 g(t_1, y(t_1)).$$

This explains the name of the method: $y(t_1)$ is only given implicitly since it occurs on both sides of the equal sign. Now we approximate the solution $y_1 \approx y(t_1)$. This yields the following algorithm: For $i = 1, \ldots, M-1$ solve

$$y_{i+1} = y_i + h_{i+1} g(t_{i+1}, y_{i+1}) \tag{2.18}$$

for y_{i+1}. In case of Burgers' equation, the function g is nonlinear and we apply a classical Newton method to solve (2.18). Reformulate (2.18) as

$$G(y_{n+1}) := y_{n+1} - y_n - h_{i+1} g(t_{n+1}, y_{n+1}).$$

with unknown y_{n+1}. The aim is to find the root y of $G(y) = 0$. Newton's algorithm is given by

1. Solve $\nabla G(u_k)\Delta u_k = -G(u_k)$ for Δu_k.
2. Update $u_{k+1} = u_k + \Delta u_k$.

We now address the different discretization techniques. For the time discretization, we use a uniform grid with fixed stepsize $\delta_t = \frac{T}{M}$, i.e. $t_j = j\delta_t$ for $j = 0, \dots, M$.

Finite Differences In case of the FD approximation the underlying system is given by

$$\dot{\alpha}(t) = \mathbf{A}_{\mathrm{FD}}\alpha(t) + \mathcal{N}_{\mathrm{FD}}\left(\alpha(t)\right) + \overline{f}(t).$$

Hence, the desired functions for the backward Euler are given as follows

$$\begin{aligned} g(u) &= \mathbf{A}_{\mathrm{FD}}u + \mathcal{N}_{\mathrm{FD}}(u) + \overline{f}, \\ G(u) &= u - y_n - \delta_t\left(\mathbf{A}_{\mathrm{FD}}u + \mathcal{N}_{\mathrm{FD}}(u) + \overline{f}\right), \\ \nabla G(u) &= \mathbf{I}_N - \delta_t\left(\mathbf{A}_{\mathrm{FD}} + \nabla\mathcal{N}_{\mathrm{FD}}(u)\right). \end{aligned}$$

Here, the gradient of the nonlinear term is given by

$$\nabla\mathcal{N}_{\mathrm{FD}}(u) = \frac{1}{2h}\begin{pmatrix} -u_2 & -u_1 & & & \\ u_2 & u_1 - u_3 & -u_2 & & \\ & \ddots & \ddots & \ddots & \\ & & u_{N-1} & u_{N-2} - u_N & -u_{N-1} \\ & & & u_N & u_{N-1} \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Finite Element Method Recall that the underlying system of the FEM is given by

$$\mathbf{M}_{\mathrm{FEM}}\dot{\alpha}(t) = \mathbf{A}_{\mathrm{FEM}}\alpha(t) + \mathcal{N}_{\mathrm{FEM}}(\alpha(t)) + \mathbf{B}_{\mathrm{FEM}}\overline{f}(t).$$

Inverting the mass matrix yields

$$\begin{aligned} g(u) &= \mathbf{M}_{\mathrm{FEM}}^{-1}\left(\mathbf{A}_{\mathrm{FEM}}u + \mathcal{N}_{\mathrm{FEM}}(u) + \mathbf{B}_{\mathrm{FEM}}\overline{f}\right), \\ G(u) &= u - y_n - \delta_t\mathbf{M}_{\mathrm{FEM}}^{-1}\left(\mathbf{A}_{\mathrm{FEM}}u + \mathcal{N}_{\mathrm{FEM}}(u) + \mathbf{B}_{\mathrm{FEM}}\overline{f}\right), \\ \nabla G(u) &= \mathbf{I}_N - \delta_t\mathbf{M}_{\mathrm{FEM}}^{-1}\left(\mathbf{A}_{\mathrm{FEM}} + \nabla\mathcal{N}_{\mathrm{FEM}}(u)\right) \end{aligned}$$

and the gradient of the nonlinear term reads as

$$\nabla\mathcal{N}_{\mathrm{FEM}}(u) = \frac{1}{6}\begin{pmatrix} -u_2 & -u_1 - 2u_2 & & & \\ 2u_1 + u_2 & u_1 - u_3 & -u_2 - 2u_3 & & \\ & \ddots & \ddots & \ddots & \\ & & 2u_{N-2} + u_{N-1} & u_{N-2} - u_N & -u_{N-1} - 2u_N \\ & & & 2u_{N-1} + u_N & u_{N-1} \end{pmatrix}.$$

Group Finite Elements In the GFE approximation, only the nonlinear term differs from the FEM implementation. Remember that it is given by $\mathcal{N}_{\text{GFE}}(u) = \mathbf{N}_{\text{GFE}}u^2$. We directly infer

$$\begin{aligned} g(u) &= \mathbf{M}_{\text{GFE}}^{-1}\left(\mathbf{A}_{\text{GFE}}u + \mathbf{N}_{\text{GFE}}u^2 + \mathbf{B}_{\text{GFE}}\overline{f}\right), \\ G(u) &= u - y_n - \delta_t\mathbf{M}_{\text{GFE}}^{-1}\left(\mathbf{A}_{\text{GFE}}u + \mathbf{N}_{\text{GFE}}u^2 + \mathbf{B}_{\text{GFE}}\overline{f}\right), \\ \nabla G(u) &= \mathbf{I}_N - \delta_t\mathbf{M}_{\text{GFE}}^{-1}\left(\mathbf{A}_{\text{GFE}} + 2\mathbf{N}_{\text{GFE}}\text{diag}(u)\right) \end{aligned}$$

where $u^2 = \begin{pmatrix} u_1^2 & u_2^2 & \cdots & u_n^2 \end{pmatrix}$ denotes the entrywise squares vector u and

$$\text{diag}(u) = \begin{pmatrix} u_1 & & & \\ & u_2 & & \\ & & \ddots & \\ & & & u_n \end{pmatrix}.$$

Implementation Neither for the FEM nor for the GFE approximation it is necessary to invert the mass matrix. Instead, we can multiply with the mass matrix $\mathbf{M}$ in step 1 to receive the equivalent problem

$$\left[\mathbf{M} + \delta_t\left(\mathbf{A} + \nabla\mathcal{N}(u_k)\right)\right]\Delta u_k = \mathbf{M}(y_n - u_k) + \delta_t\left(\mathbf{A}u_k + \mathcal{N}(u_k) + \mathbf{B}\overline{f}\right).$$

This is more feasible regarding the computational costs. To initialize Newton's method we use one iteration of the forward (explicit) Euler, that is

$$u_1 = y_n + \delta_t\mathbf{M}^{-1}\left(\mathbf{A}y_n + \mathcal{N}\left(y_n\right) + \mathbf{B}\overline{f}\right).$$

These aspects result into Algorithm 1.

2.5 Test Problems

As outlined in the introduction one of the main reasons to investigate Burgers' equation as test equation for numerical algorithms is the fact that one can find analytical solutions for a specific problem setting. Beside the Hopf-Cole transformation we can also use the Method of Manufactured Solutions (MMS) ([58], [59], [60], [62], [50]) to generate analytical solutions. In this thesis we employ two examples. The first is based on the MMS with a forcing term and is used to validate the presented methods. The second example investigates a discontinuous initial condition and is borrowed from [44].

Example 1 (Method of Manufactured Solutions) The MMS is based on a free choice of the forcing term. Let $\mathcal{D}$ denote the differential operator associated with Burgers' equation, that is

$$\mathcal{D}y = y_t + yy_x - y_{xx}$$

Algorithm 1 Implicit Euler

Input: Number of time steps M, stepsize δ_t, dynamical system $\dot{y}(t) = g(t, y(t))$, desired error in Newton's algorithm err.
Output: Solution of the dynamical system.

Let y_1 denote the initial condition. For $i = 2, \ldots, M$

1. Compute initialization for Newton's method with the forward Euler

$$u_0 = y_{i-1} + \delta_t g(t_{i-1}, y_{i-1})$$

   and define $G(w) = w - y_{i-1} + \delta_t g(t_i, w)$.

2. Set $k = 0$.

3. While $\|G(u_k)\|_2 > err$

   (a) Compute the gradient $\nabla G(u_k)$.

   (b) Solve

$$\nabla G(u_k)\Delta u_k = -G(u_k) \qquad \text{for } \Delta u_k$$

   (c) Update: $u_{k+1} = u_k + \Delta u_k, k = k + 1$.

4. Set $y_i = u_k$.

for suitable y. We construct an exact solution of (2.1) by finding a function $\overline{y}$ that meets the boundary condition and set the initial condition as $y_0(x) := \overline{y}(0, x)$. To satisfy (2.1a) we set the forcing term to $f(t, x) = \mathcal{D}y(t, x)$. We consider the example given in [38], which is based on the function

$$y(t, x) = e^{-t} \sin(\pi x).$$

Then the initial condition becomes $y(0, x) = \sin(\pi x) = y_0(x)$ and it is easy to see that the boundary conditions are satisfied. The associated MMS forcing term is then given by

$$f(t, x) = -e^{-t} \sin(\pi x) + \pi e^{-2t} \sin(\pi x) \cos(\pi x) + \nu \pi^2 e^{-t} \sin(\pi x).$$

Note that the solution $y(t, x)$ is independent of the choice of the viscosity parameter ν. The MMF solution is presented in Figure 2.2.

Example 2 (Initial Shock) As outlined in Section 1.3, the viscid Burgers' equation turns into the inviscid Burgers' equation with growing Reynolds number. The method of characteristics infers - after some time - a discontinuous solution even for a smooth initial condition. In theory, this is called a shock. We simulate this fact by using a shock initial condition in our second example, which is borrowed from [44]:

$$y_0(x) = \begin{cases} 1, & \text{if } x \in \left[0, \frac{1}{2}\right] \\ 0, & \text{otherwise.} \end{cases}$$

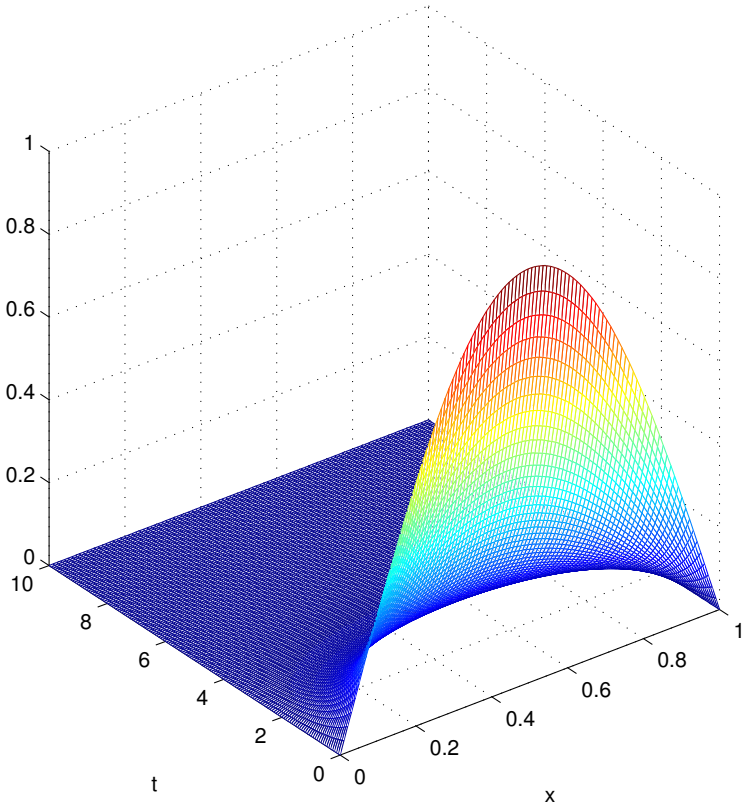


Figure 2.2: MMF solution of Burgers' equation

We use the MATLAB® function PDEPE to get a reference solution, which can be used for error computations. The reference solution is displayed in Figure 2.3

2.6 Computation of the L^2 norm

In the following we derive a tool for error measurements. A reasonable choice is the common L^2 norm. The solution of Example 1 is $y(t,x) = e^{-t}\sin(\pi x)$ and hence, the L^2 norm computes to

$$\begin{aligned} \|y\|_{L^2(\Omega)} &= \left(\int_0^T \int_0^1 e^{-2t}\sin^2(\pi x)dxdt\right)^{\frac{1}{2}} = \left(\int_0^T e^{-2T}dt \int_0^1 \sin^2(\pi x)dx\right)^{\frac{1}{2}} \\ &= \sqrt{\frac{1}{4}(1-e^{-2T})}. \end{aligned}$$

However, this is not very helpful to measure the error in the numerical computations. Instead, the goal of this section is to derive an approximation of the L^2 norm. We employ the trapezoidal rule to approximate the integrals

$$\int_a^b g(x)dx \approx \frac{b-a}{2}\left(g(a)+g(b)\right). \tag{2.19}$$

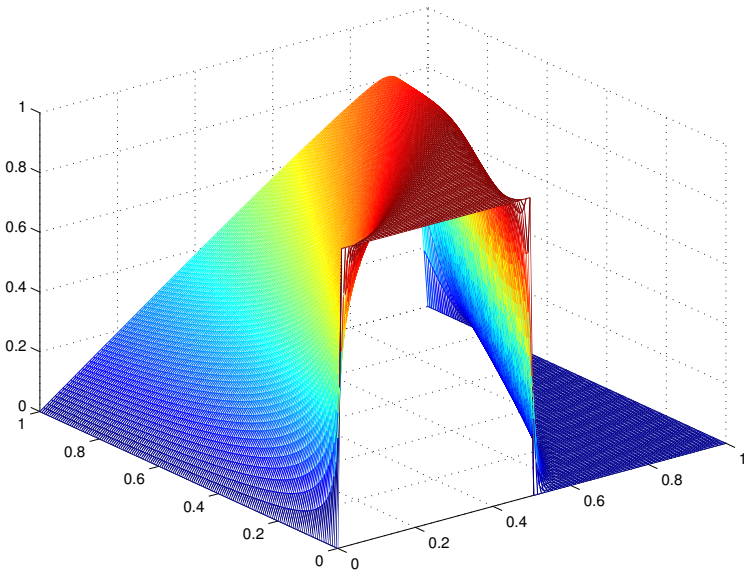


Figure 2.3: Reference solution for Example 2

Since our grid is uniform with step size h, (2.19) becomes

$$\int_0^1 g(x)dx \approx \frac{h}{2}\left(g\left(x_1\right) + g\left(x_N\right) + 2\sum_{k=2}^{N-1} g\left(x_k\right)\right),$$

where we took advantage of the Dirichlet boundary conditions (2.1b) and omitted the points x_0 and x_{N+1} in the spatial grid. Introducing the matrix $\mathbf{T}_s \in \mathbb{R}^{N\times N}$

$$\mathbf{T}_s = \begin{pmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & \frac{1}{2} \end{pmatrix} \in \mathbb{R}^{N\times N}$$

yields

$$\int_0^1 |y(t,x)|^2 dx \approx \overline{y}(t)^T \mathbf{T}_s \overline{y}(t)$$

for the spatial approximation. Similarly, for the approximation in time we use the matrix $\mathbf{T}_t \in \mathbb{R}^{M\times M}$ given by

$$\mathbf{T}_t = \begin{pmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & \frac{1}{2} \end{pmatrix} \in \mathbb{R}^{M\times M},$$

which equals the spatial approximation matrix, not taking the dimension into account. Again, this approach offers a computationally feasible implementation in MATLAB® .

2.7 Numerical Results

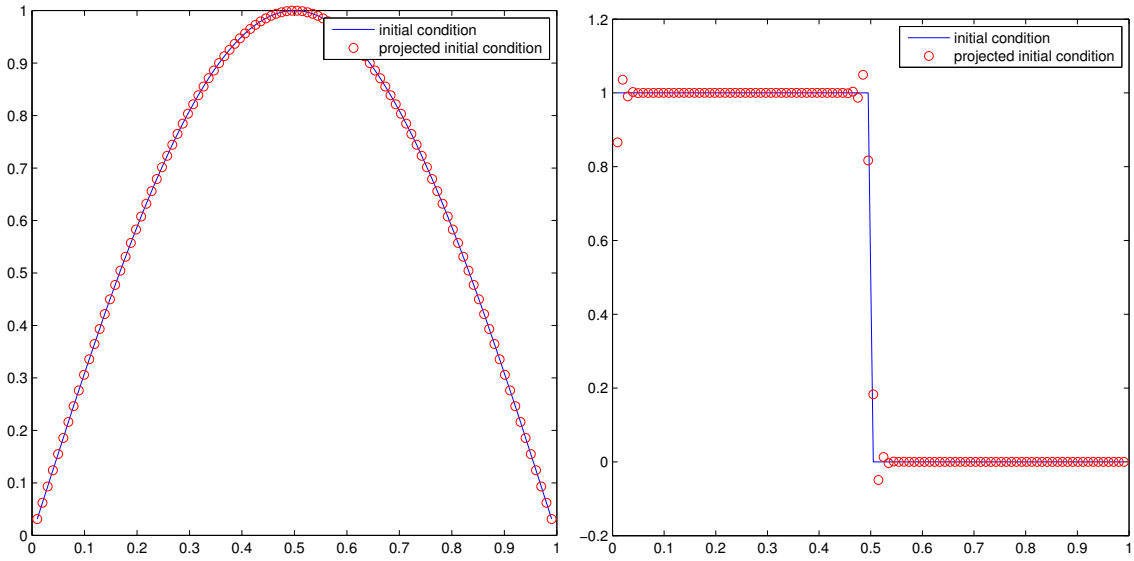
We demonstrate the accuracy of the presented methods by solving the test problems introduced in Section 2.5. For detailed information about the computational resources used throughout the numerical study see Table 2.1.

OS	Processor	Memory
Mac OS X, Version 10.8.4	1.3 GHz Intel Core i5	4 GB 1600 MHz DDR 3

Software	Machine Epsilon
MATLAB® R2013a (8.1.0.604), 64 bit	2.2204e-16

Table 2.1: Computational environment

We validate the projected initial condition for both examples. The original and projected initial condition for FEM (and hence also for GFEM) for $N = 100$ spatial discretization points are presented in Figure 2.4. Note that for Example 1, the projected initial condition



(a) Example 1 (b) Example 2

Figure 2.4: Projected initial condition

almost exactly matches the initial condition, while the approximation for Example 2 is not as close to the discontinuity. The error in the standard Euclidean norm in $\mathbb{R}^N$ for both examples is given in Table 2.2. The discontinuity has no representation in the FE space. This implies that even for larger N, something similar to the Gibbs phenomena appears at the discontinuity. However, in our numerical computations we use the original initial condition instead of the projected one. Regarding Example 1, the error is of order 10^{-5}

Example 1:	0.000081
Example 2:	0.042780

Table 2.2: Relative Euclidean error of the projected initial condition

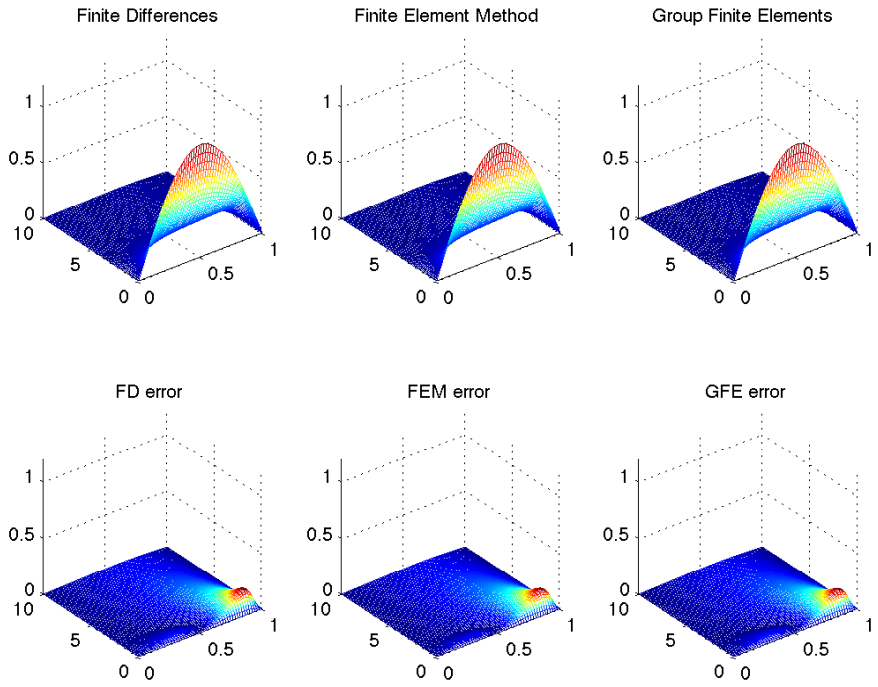


Figure 2.5: Example 1: Approximate solutions and absolute errors with backward Euler

which is very small compared to the error obtained by the discretization process. Regarding Example 2, the error is significantly greater and we can not represent the discontinuity in a reasonable fashion. Hence, the not projected initial condition yields "better" results.

For the following computations we use $\nu = 1/100$ as reciprocal of the Reynolds numbers. Other researchers, for example Pugh [57], Smith [65] and Krämer [43], have investigated different Reynolds numbers, for which reason additional computations with further Reynolds numbers are not necessary. In Example 1, we employ $T = 10$ in contrast to $T = 1$ in Example 2. Recall that the build-in solvers of MATLAB® have an adaptive step size control. Hence, they choose their own value of M. For our own implementation of the backward Euler we use twice the number of discretization points, that is $M = 2N$.

We start the discussion with solution plots for Example 1. We present the approximate solution and absolute error obtained by the backward Euler in Figure 2.5, while Figure 2.6 presents the same results employing the stiff MATLAB® solver ODE15S.

Obviously it is not possible to see any difference between the discretization techniques in Figure 2.5 and Figure 2.6. However, the structure of the error differs between the solvers. The main reason is the adaptive step size in the MATLAB® solver ODE15S compared to

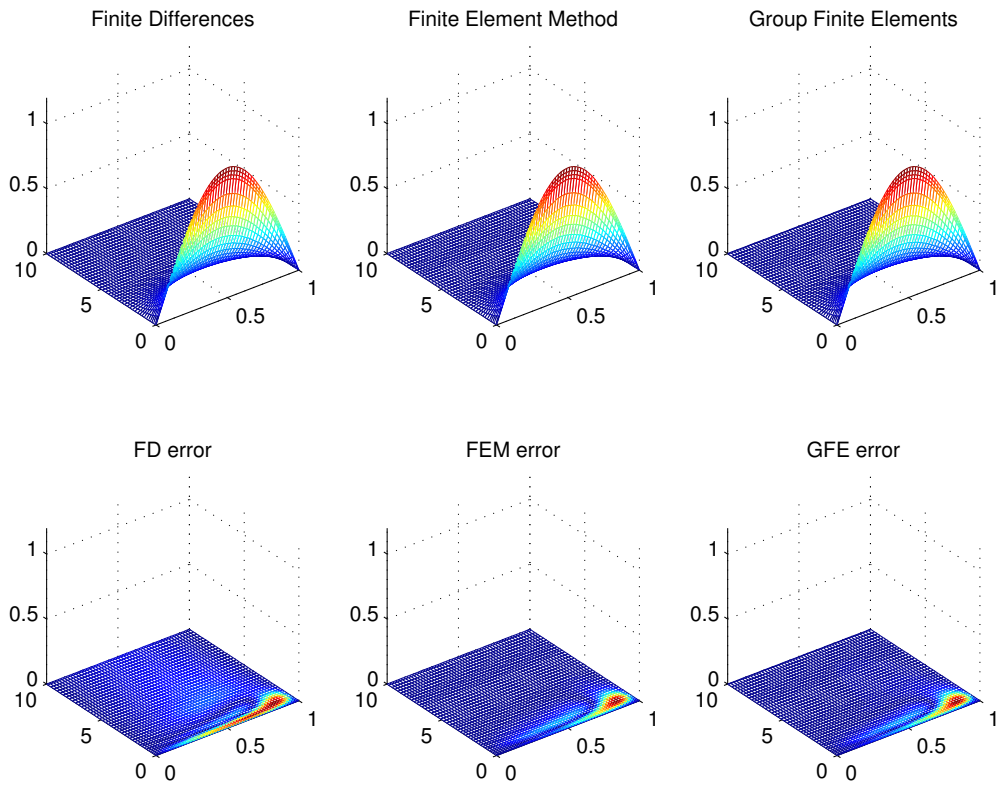


Figure 2.6: Example 1: Approximate solutions and absolute errors with the MATLAB® solver ODE15S

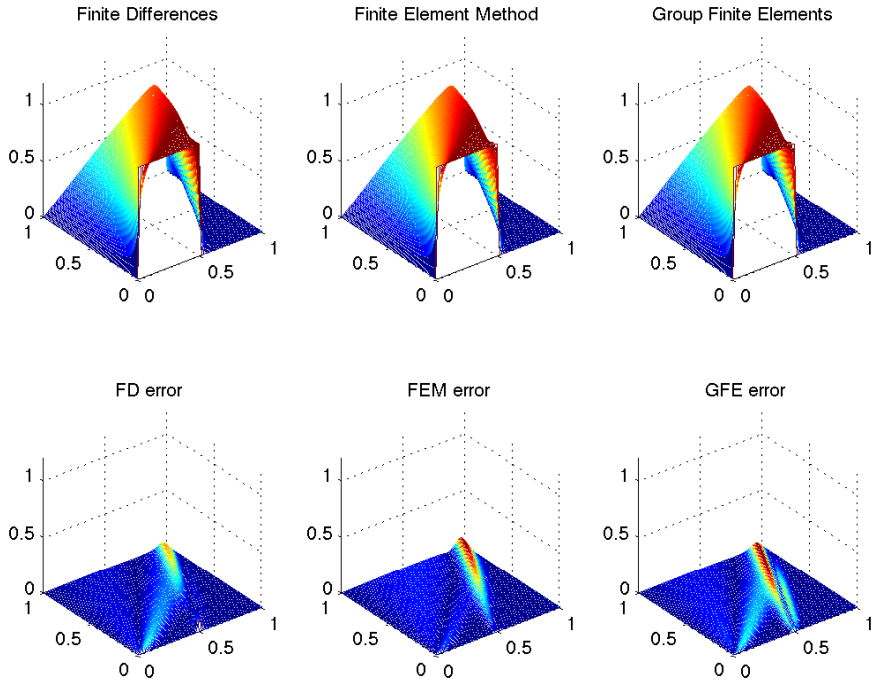


Figure 2.7: Example 2: Approximate solutions and absolute errors with backward Euler

equidistant time steps in the backward Euler. In Table 2.3 we provide the relative errors for the discretization techniques and solvers for varying numbers of N. Comparing the data for the backward Euler we observe the following. For all three discretization methods, the error decays with increasing number of discretization points N. This is exactly the phenomenon we expected. The finer the discretization, the better the approximation. Moreover, the GFE outperforms both other techniques for all numbers of N. This coincides with the results of [57] and [65]. For $N = 10$, the FEM yields a better approximation than the FD approach, but for $N \geq 30$ the convers holds. However, for $N \geq 110$ all three discretization techniques provide a similar error less than 1%.

The situation is slightly different if we employ some MATLAB® solver. Among these three solver, the FD approach creates the best approximations. All computed relative errors are less than 1% and decaying very fast. The best results are produced by the ODE45 solver. An interesting phenomenon appears with the ODE15S solver. The errors are not decaying with increasing N and for $N = 190$ the error is almost the same (FEM, GFE) as for $N = 30$ while it is even worse in the FD case.

We observe a similar behavior in the computations of Example 2, displayed in Figure 2.7 and Figure 2.8. This time, we can observe different error plots for the discretization techniques.

Common for all discretization techniques is a significant error along the shock. Due to the viscosity term y_{xx} this is not a discontinuity, but still, all three discretization techniques struggle with this part. The adaptive method ODE15S handles the issue much better than the backward Euler and yet there is a significant error along the shock. The relative errors are presented in Table 2.4. Almost all the time, GFE provides the best results closely followed

		FD	FEM	GFE
backward Euler	$N = 10$	0.660749	0.435186	0.369630
	$N = 30$	0.090067	0.091911	0.086522
	$N = 50$	0.039739	0.041113	0.039210
	$N = 70$	0.022690	0.023547	0.022548
	$N = 90$	0.014740	0.015311	0.014687
	$N = 110$	0.010365	0.010769	0.010339
	$N = 130$	0.007693	0.007992	0.007676
	$N = 150$	0.005939	0.006169	0.005926
	$N = 170$	0.004725	0.004906	0.004713
	$N = 190$	0.003849	0.003995	0.003838
ode45	$N = 10$	0.000157	0.000809	0.000308
	$N = 30$	0.000002	0.000013	0.000006
	$N = 50$	0.000000	0.000002	0.000001
	$N = 70$	0.000000	0.000000	0.000000
	$N = 90$	0.000000	0.000000	0.000000
	$N = 110$	0.000000	0.000000	0.000000
	$N = 130$	0.000000	0.000000	0.000000
	$N = 150$	0.000000	0.000000	0.000000
	$N = 170$	0.000000	0.000000	0.000000
	$N = 190$	0.000000	0.000000	0.000000
ode15s	$N = 10$	0.000154	0.000824	0.000329
	$N = 30$	0.000004	0.000022	0.000013
	$N = 50$	0.000003	0.000007	0.000006
	$N = 70$	0.000003	0.000005	0.000004
	$N = 90$	0.000003	0.000004	0.000004
	$N = 110$	0.000003	0.000004	0.000004
	$N = 130$	0.000003	0.000004	0.000003
	$N = 150$	0.000002	0.000002	0.000002
	$N = 170$	0.000003	0.000003	0.000003
	$N = 190$	0.000016	0.000016	0.000015
ode23s	$N = 10$	0.000135	0.000874	0.000351
	$N = 30$	0.000002	0.000023	0.000013
	$N = 50$	0.000001	0.000007	0.000005
	$N = 70$	0.000001	0.000004	0.000003
	$N = 90$	0.000002	0.000003	0.000002
	$N = 110$	0.000002	0.000002	0.000002
	$N = 130$	0.000002	0.000002	0.000002
	$N = 150$	0.000002	0.000002	0.000002
	$N = 170$	0.000002	0.000002	0.000002
	$N = 190$	0.000002	0.000002	0.000002

Table 2.3: Example 1: Relative error for different discretization techniques, solver and discretization points N

		FD	FEM	GFE
backward Euler	$N = 10$	0.015773	0.013137	0.011280
	$N = 30$	0.000986	0.002225	0.001036
	$N = 50$	0.000395	0.001039	0.000408
	$N = 70$	0.000226	0.000607	0.000227
	$N = 90$	0.000148	0.000399	0.000147
	$N = 110$	0.000105	0.000283	0.000103
	$N = 130$	0.000079	0.000211	0.000077
	$N = 150$	0.000061	0.000163	0.000059
	$N = 170$	0.000049	0.000129	0.000046
	$N = 190$	0.000040	0.000106	0.000038
ode45	$N = 10$	0.056976	0.025940	0.016235
	$N = 30$	0.001444	0.000953	0.000686
	$N = 50$	0.000191	0.000174	0.000128
	$N = 70$	0.000053	0.000055	0.000040
	$N = 90$	0.000021	0.000023	0.000016
	$N = 110$	0.000010	0.000011	0.000008
	$N = 130$	0.000005	0.000006	0.000004
	$N = 150$	0.000003	0.000004	0.000003
	$N = 170$	0.000002	0.000002	0.000002
	$N = 190$	0.000001	0.000002	0.000001
ode15s	$N = 10$	0.057073	0.025935	0.016237
	$N = 30$	0.001451	0.000953	0.000686
	$N = 50$	0.000191	0.000175	0.000128
	$N = 70$	0.000053	0.000055	0.000040
	$N = 90$	0.000021	0.000023	0.000017
	$N = 110$	0.000010	0.000011	0.000008
	$N = 130$	0.000005	0.000006	0.000004
	$N = 150$	0.000003	0.000004	0.000003
	$N = 170$	0.000002	0.000002	0.000002
	$N = 190$	0.000001	0.000002	0.000001
ode23s	$N = 10$	0.057102	0.025950	0.016220
	$N = 30$	0.001461	0.000952	0.000685
	$N = 50$	0.000194	0.000174	0.000127
	$N = 70$	0.000055	0.000055	0.000039
	$N = 90$	0.000022	0.000022	0.000016
	$N = 110$	0.000010	0.000011	0.000008
	$N = 130$	0.000006	0.000006	0.000004
	$N = 150$	0.000003	0.000004	0.000002
	$N = 170$	0.000002	0.000002	0.000002
	$N = 190$	0.000001	0.000001	0.000001

Table 2.4: Example 2: Relative error for different discretization techniques, solver and discretization points N

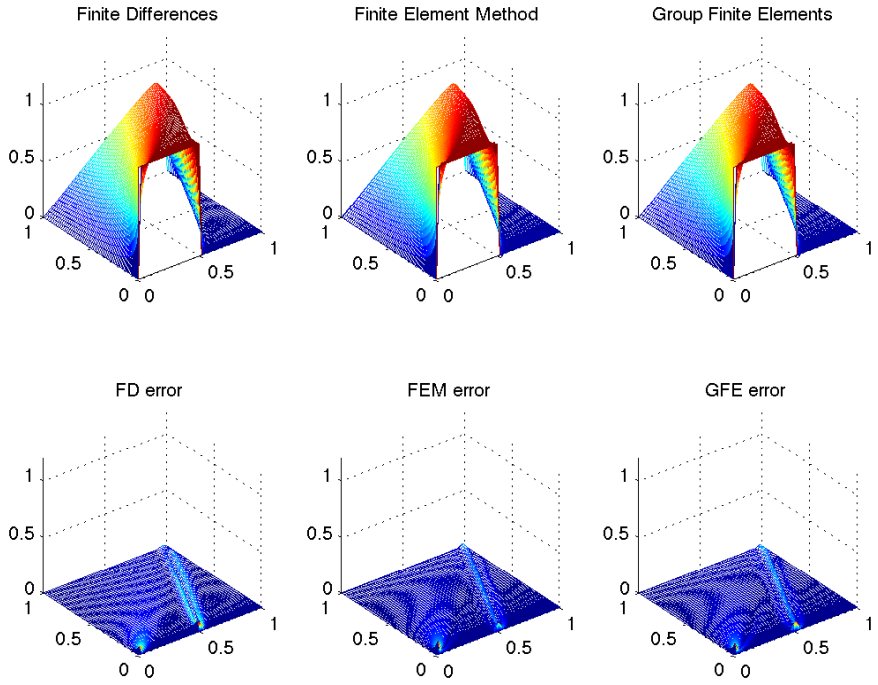
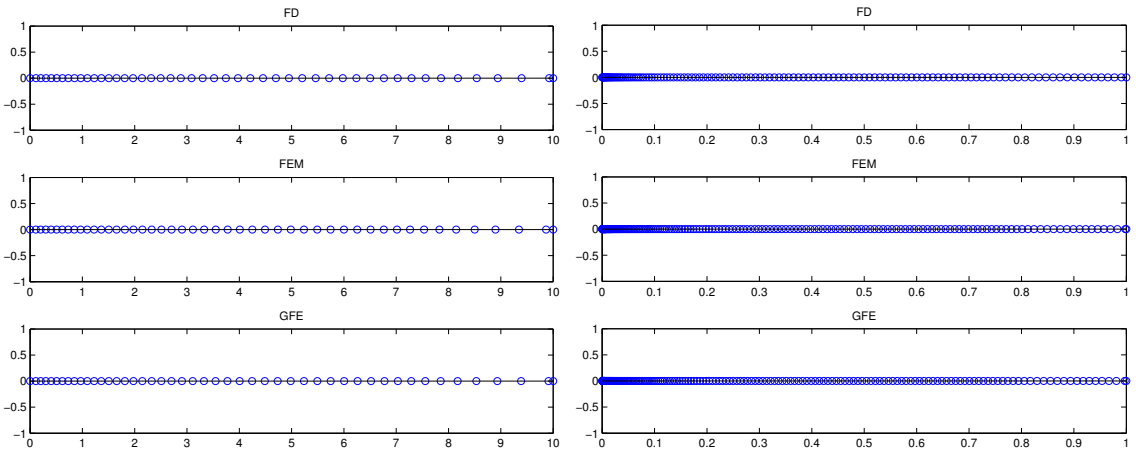


Figure 2.8: Example 2: Approximate solutions and absolute errors with the MATLAB® solver ODE15S

by the FD approximation. However, the error is not decaying as fast as in Example 1 in terms of the MATLAB® solvers, while the results of the backward Euler are of order 10^2 better compared with its results in Example 1. For small numbers of N, the backward Euler provides even better results than the adaptive solver. All three MATLAB® solver result in very similar approximations.

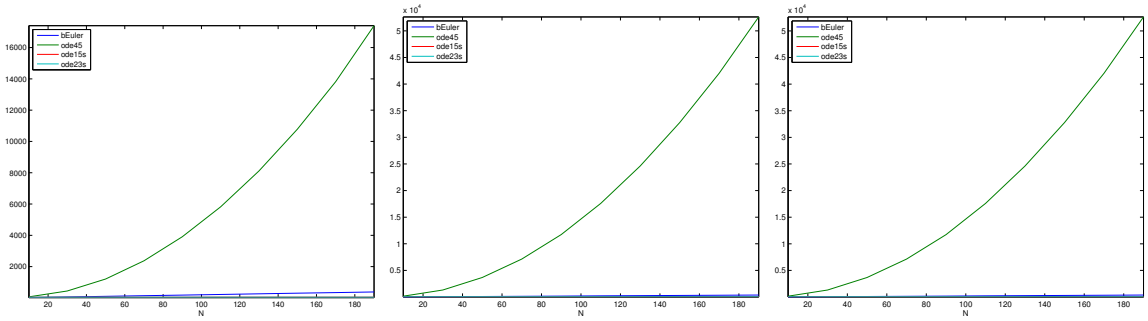
To understand the error development of the different solvers, we investigate the adaptive time grid they used for the computations. In Figure 2.9 we present the distribution of the temporal grid points for both examples. For both examples, the time grid is finer at the initial time point ($t = 0$) and gets more coarse to the final time point ($t = T$). The next step is a more detailed analysis of the time grids. For example uses the ODE45 solver much more temporal discretization points than the other employed solver, as we can see in Figure 2.10 and Figure 2.11. Moreover, the FD approximation gets along with less temporal discretization points, than FEM and GFE and the second example also needs less points than the first one. As we have seen in Figure 2.9, the smallest time steps are close to the initial time point. This leads naturally to the question of the evolution of the minimal step size. Figure 2.12 and Figure 2.13 illustrate this question. While for Example 1 the number of spatial discretization points N seems to have no effect on the stiff solvers ODE15S and ODE23S, they decay similar as the non stiff solver ODE45 in Example 2. Note that we have $M = 2N$ equally distributed temporal discretization points for the implementation of the backward Euler.

We conclude the discussion of the numerical results with some time measurements for the



(a) Example 1 (b) Example 2

Figure 2.9: Distribution of temporal grid points ($N = 50$, ODE23)

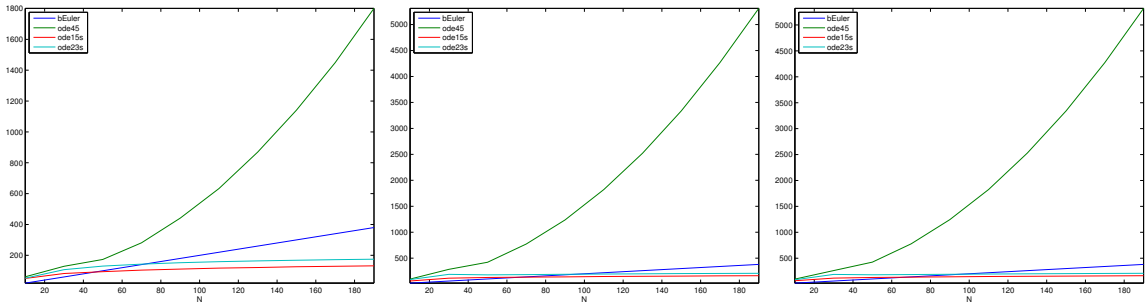


(a) Finite Differences (b) FEM (c) GFE

Figure 2.10: Example 1: Number of temporal grid points

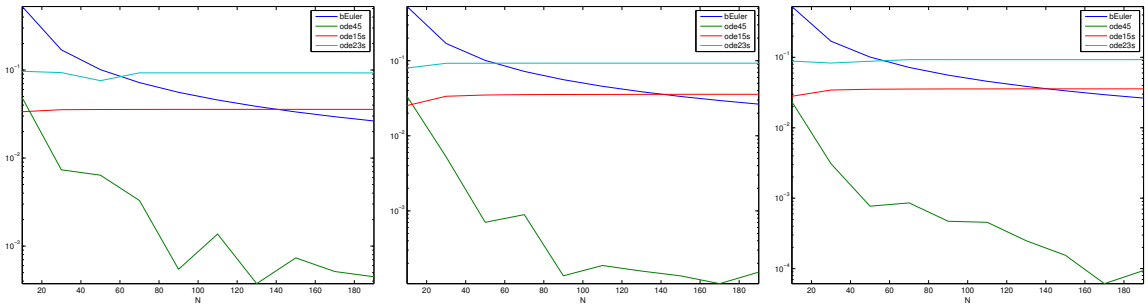
four solver. In order to receive resilient results, the measured time is the average over 50 reruns of the computation. We only consider the time used to integrate the ODE, which we refer to as *online cost*, without the time to assemble the discretization matrices (*offline cost*). The averaged computational times are presented in Table 2.5 and Table 2.6 for Example 1 and Example 2 respectively.

Similar to the relative error, GFE outperforms both other discretization techniques if the backward Euler is employed. In terms of the MATLAB® solvers, GFEM still yields better results the the common FEM. However, this time, the FD approximation is computed faster with the MATLAB® solvers. For both examples, ODE15S is the fastest solver. For Example 1, ODE45 is the slowest solver, in Example 2 ODE23S takes this spot. The computational effort for the backward Euler is not as good as the best solver, but still has a good performance and is hence, a reasonable choice, if for example the forcing term is only available at fixed time points, which is the case in the optimal control setting in Chapter 4.



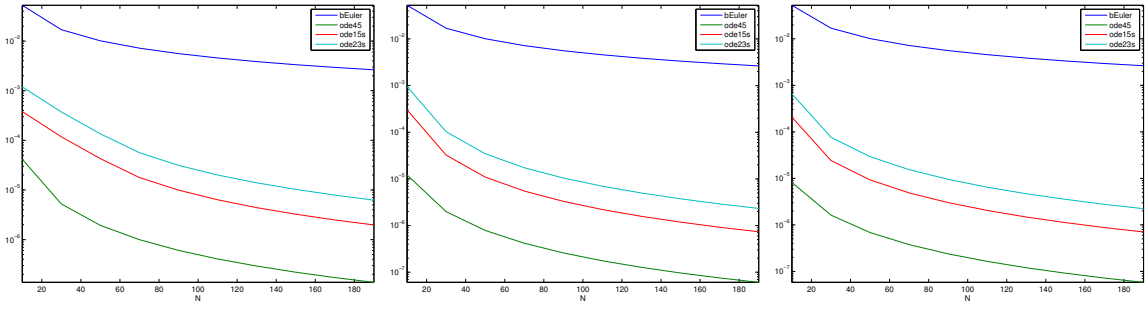
(a) Finite Differences (b) FEM (c) GFE

Figure 2.11: Example 2: Number of temporal grid points



(a) Finite Differences (b) FEM (c) GFE

Figure 2.12: Example 1: Minimal time step



(a) Finite Differences (b) FEM (c) GFE

Figure 2.13: Example 2: Minimal time step

		FD	FEM	GFE
backward Euler	$N = 10$	0.006178	0.007372	0.005773
	$N = 30$	0.019267	0.021072	0.015219
	$N = 50$	0.034284	0.037892	0.030322
	$N = 70$	0.060692	0.070065	0.050115
	$N = 90$	0.087776	0.101121	0.075675
	$N = 110$	0.126386	0.150885	0.112186
	$N = 130$	0.199486	0.235411	0.172020
	$N = 150$	0.272639	0.314982	0.237165
	$N = 170$	0.358028	0.408190	0.304419
	$N = 190$	0.433545	0.496415	0.370595
ode45	$N = 10$	0.019414	0.043635	0.039402
	$N = 30$	0.079616	0.327854	0.295651
	$N = 50$	0.216568	0.930398	0.830048
	$N = 70$	0.449264	1.927240	1.759783
	$N = 90$	0.797838	3.270078	3.054425
	$N = 110$	1.237571	5.154435	4.740713
	$N = 130$	1.799756	7.446322	6.903528
	$N = 150$	2.524612	10.326094	9.575569
	$N = 170$	3.365499	13.765659	12.773581
	$N = 190$	4.144271	16.913115	15.911294
ode15s	$N = 10$	0.030942	0.037629	0.028944
	$N = 30$	0.034782	0.038029	0.037260
	$N = 50$	0.046422	0.052176	0.051797
	$N = 70$	0.061340	0.064465	0.060473
	$N = 90$	0.066065	0.075138	0.071206
	$N = 110$	0.076813	0.085373	0.080593
	$N = 130$	0.090507	0.100539	0.094018
	$N = 150$	0.102889	0.117972	0.106535
	$N = 170$	0.142139	0.159140	0.148454
	$N = 190$	0.158513	0.172833	0.160408
ode23s	$N = 10$	0.074185	0.086530	0.074069
	$N = 30$	0.156839	0.184310	0.164568
	$N = 50$	0.301818	0.351503	0.314236
	$N = 70$	0.437741	0.498129	0.446432
	$N = 90$	0.583630	0.667610	0.591375
	$N = 110$	0.742060	0.859667	0.765185
	$N = 130$	0.916559	1.045316	0.943606
	$N = 150$	1.117247	1.265327	1.147702
	$N = 170$	1.310371	1.462036	1.297691
	$N = 190$	1.497630	1.675651	1.543865

Table 2.5: Example 1: Computational time for different discretization techniques, solver and discretization points N

		FD	FEM	GFE
backward Euler	$N = 10$	0.005780	0.007477	0.005044
	$N = 30$	0.021570	0.022321	0.015637
	$N = 50$	0.033429	0.044226	0.031372
	$N = 70$	0.055249	0.079824	0.054004
	$N = 90$	0.085128	0.121993	0.084342
	$N = 110$	0.127385	0.181246	0.125099
	$N = 130$	0.203843	0.289829	0.199275
	$N = 150$	0.279771	0.353989	0.243359
	$N = 170$	0.384007	0.364278	0.250830
	$N = 190$	0.512325	0.419162	0.293267
ode45	$N = 10$	0.014278	0.025082	0.020924
	$N = 30$	0.020891	0.066070	0.053328
	$N = 50$	0.029504	0.105399	0.092967
	$N = 70$	0.048570	0.182509	0.170936
	$N = 90$	0.072732	0.300545	0.277849
	$N = 110$	0.105860	0.441836	0.398207
	$N = 130$	0.141880	0.611622	0.549814
	$N = 150$	0.191467	0.813437	0.734358
	$N = 170$	0.243148	1.049783	0.951163
	$N = 190$	0.305657	1.316129	1.186956
ode15s	$N = 10$	0.025951	0.034401	0.029180
	$N = 30$	0.039893	0.055643	0.051760
	$N = 50$	0.054141	0.070523	0.067011
	$N = 70$	0.063375	0.082017	0.078082
	$N = 90$	0.072185	0.092843	0.090415
	$N = 110$	0.082680	0.101318	0.098559
	$N = 130$	0.093917	0.117578	0.111488
	$N = 150$	0.106862	0.128152	0.125395
	$N = 170$	0.123706	0.147960	0.139979
	$N = 190$	0.131597	0.163094	0.155019
ode23s	$N = 10$	0.077287	0.148823	0.126168
	$N = 30$	0.318468	0.659505	0.582110
	$N = 50$	0.657842	1.080685	0.956487
	$N = 70$	0.990305	1.544303	1.340967
	$N = 90$	1.340400	2.031449	1.767625
	$N = 110$	1.713910	2.532207	2.203784
	$N = 130$	2.128640	3.083347	2.709037
	$N = 150$	2.618317	4.378293	3.279221
	$N = 170$	2.988400	4.321747	3.749782
	$N = 190$	3.458303	4.959259	4.339347

Table 2.6: Example 2: Computational time for different discretization techniques, solver and discretization points N

Chapter 3

Reduced Order Modeling (ROM)

A challenge in numerical computations concerning PDE is obtaining high accuracy while solving the resulting large-scale dynamical systems as fast as possible. In this thesis, we investigate the method of *Reduced Order Modeling* (ROM) also known as *Model Order Reduction* (MOR) [18]. The goal of this method is to find a low dimensional model of the high dimensional system. Various techniques have been investigated over the last two decades, for example Balanced Truncation (BT), Tangential Interpolation and Proper Orthogonal Decomposition (POD). The interested reader is referred to [4], [27] and [3] for more details.

In this work, we focus on POD and use the Discrete Empirical Interpolation Method (DEIM) introduced by Chaturantabut [17], [18], [19], [20] to improve the computational effort for the nonlinear term in Burgers' equation. We start the discussion of ROM with POD in Section 3.1 with focus on the structure of the POD basis. It follows the derivation of the reduced-order model with POD in Section 3.2. Similar to GFEM introduced in Section 2.3, a Group POD (GPOD) method introduced by Dickinson [23] is discussed in Section 3.3. DEIM is investigated in Section 3.4. The chapter concludes with numerical results.

3.1 Proper Orthogonal Decomposition (POD)

POD is probably the most studied reduction technique in terms of reducing nonlinear systems over the last two decades. It was developed independently by several researchers and is also known as *Principal Component Analysis* (PCA), *Method of Snapshots*, and *Karhunen-Loéve Decomposition*. Good overviews of the evolution of POD and its application in different research areas are provided in [31] and [34]. To introduce POD, we recall the dynamical system

$$\mathbf{M}\dot{\alpha}(t) = \mathbf{A}\alpha(t) + \mathcal{N}(\alpha(t)) + \mathbf{B}\overline{f}(t), \tag{3.1}$$

derived in Chapter 2. Note that although this form was specifically designed as discretized version of Burgers' equation, many discretized PDEs result in the same form. Hence, the discussion of POD is not limited to Burgers' equation but directly extends to a wide class of

dynamical systems. POD is, as many other reduction techniques, a projection based technique. In particular, a Galerkin projection is used to determine the low rank approximation of the full system. Analogue to the previous chapter, assume that (3.1) is an N-dimensional dynamical system, and let $\ell \in \mathbb{N}$ denote the desired dimension of the reduced system. Suppose that $\mathbf{V}_\ell \in \mathbb{R}^{N\times\ell}$ is an $\mathbf{M}$-orthogonal matrix, that is $\mathbf{V}_\ell^T\mathbf{M}\mathbf{V}_\ell = \mathbf{I}_\ell \in \mathbb{R}^{\ell\times\ell}$. Assume that α lives in the subspace spanned by $\mathbf{V}_\ell$.s Let

$$\alpha = \mathbf{V}_\ell \alpha_r \qquad \text{for some} \qquad \alpha_r \in \mathbb{R}^\ell. \tag{3.2}$$

Substituting (3.2) in (3.1) yields

$$\mathbf{M}\mathbf{V}_\ell \dot{\alpha}_r(t) = \mathbf{A}\mathbf{V}_\ell \alpha_r(t) + \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) + \mathbf{B}\overline{f}(t). \tag{3.3}$$

The system (3.3) is overdetermined and has no solution in general since $\ell < N$. A common approach is to project the system in an ℓ-dimensional subspace $\mathbf{U}_\ell$. The method is called *Petrov-Galerkin projection*. In case of POD, we use the projection $\mathbf{U}_\ell = \mathbf{V}_\ell$. We take the Euclidean inner product of (3.3) with $\mathbf{V}_\ell$ and infer the dynamical system

$$\dot{\alpha}_r(t) = \mathbf{V}_\ell^\top \mathbf{A}\mathbf{V}_\ell \alpha_r(t) + \mathbf{V}_\ell^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) + \mathbf{V}_\ell^\top \mathbf{B}\overline{f}(t). \tag{3.4}$$

Note that $\mathbf{V}_\ell^\top \mathbf{M}\mathbf{V}_\ell = \mathbf{I}_\ell$. Furthermore, we can precompute the matrices $\mathbf{A}_r = \mathbf{V}_\ell^\top \mathbf{A}\mathbf{V}_\ell \in \mathbb{R}^{\ell\times\ell}$ and $\mathbf{B}_r = \mathbf{V}_\ell^\top \mathbf{B} \in \mathbb{R}^{\ell\times N}$, and there is no additional cost in the evaluation of (3.4). Using only one projection matrix is a special case of the Petrov-Galerkin projection and commonly called *Galerkin Projection*. Suppose $\tilde{\alpha}(t)$ solves (3.4). For reasonable $\mathbf{V}_\ell$, a approximation of the full system (3.1) is given by

$$\alpha(t) \approx \mathbf{V}_\ell \alpha_r(t).$$

Definition 3.1 (Low-dimensional approximation)**.** *For $\ell \ll N$ the ℓ-dimensional system* (3.4) *is a* low-dimensional approximation *for* (3.1) *and therefore called a* reduced-order model.

The next section comments on how to find the projection matrix $\mathbf{V}_\ell$, which is given by the POD basis.

3.1.1 POD Basis

Both, POD and FEM are based on a Galerkin projection. In contrast to FEM, which uses local basis functions $\{\phi_i\}$, POD involves global basis functions $\{\phi_i^P\}$, which reflect the underlying dynamics of the system. More precisely, let $y^j(x) = y(t_j, x)$ be the solution of the system (3.1) at time points t_j (j=1,...,M). Note that this information is available if the system is precompute using the FEM outlined in Section 2.2.

Definition 3.2 (Snapshot set)**.** *We call $y^j(x)$* snapshots *and define the* snapshot set

$$\mathcal{Y}^{SNAP} = \{y^1, \ldots, y^m\}.$$

The snapshot matrix *is given by* $\boldsymbol{Y}^{SNAP} = \begin{pmatrix} y^1 & \cdots & y^m \end{pmatrix}$.

Extracting the most important information of the snapshot set in terms of ℓ linearly independent basis vectors $\{\phi_i^P\}_{i=1}^{\ell}$, means solving the following optimization problem:

$$\min_{\{\phi_i^P\}\subseteq\mathbb{R}^N} \sum_{j=1}^{m} \|y^j - \sum_{i=1}^{\ell} \langle y^j, \phi_i^P\rangle_{\mathbb{R}^N} \phi_i^P\|_{\mathbb{R}^n}^2 \qquad \text{such that} \qquad \langle \phi_i^P, \phi_j^P\rangle_{\mathbb{R}^N} = \delta_{i,j}. \tag{3.5}$$

Applying the common Fourier decomposition, Pythagoras' theorem and Parseval's identity, (3.5) is equivalent to

$$\max_{\{\phi_i^P\}\subseteq\mathbb{R}^N} \sum_{j=1}^{m} \sum_{i=1}^{\ell} |\langle y^j, \phi_i^P\rangle_{\mathbb{R}^N}|^2 \qquad \text{such that} \qquad \langle \phi_i^P, \phi_j^P\rangle_{\mathbb{R}^N} = \delta_{i,j}. \tag{3.6}$$

Here, equivalent means that if $\{\phi_i^P\}_{i=1}^{\ell}$ solves (3.5) then it solves (3.6), and the converse holds as well. Following [68], we introduce the Lagrange functional $\mathcal{L} : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}$ by

$$\mathcal{L}(\phi_1^P, \lambda) = \sum_{j=1}^{m} |\langle y^j, \phi_1^P\rangle_{\mathbb{R}^N}|^2 + \lambda \left(\|\phi_1^P\|_{\mathbb{R}^N}^2 - 1\right)$$

in order to solve (3.6) for $\ell = 1$. The first order optimality condition is given by

$$\nabla_{\phi_1^P} \mathcal{L} \overset{!}{=} 0 \qquad \text{and} \qquad \nabla_{\lambda} \mathcal{L} \overset{!}{=} 0.$$

According to [68], the solution of (3.6) is the set of the ℓ leading left singular vectors of $\mathbf{Y}^{\text{SNAP}}$. This leads to the following definition:

Definition 3.3 (POD basis of rank ℓ)**.** *The ℓ leading left singular vectors of $\boldsymbol{Y}^{SNAP}$ are called* POD basis of rank ℓ *and are denoted with $\{\phi_i^P\}_{i=1}^{\ell}$.*

The close relation to the singular value decomposition naturally follows from Theorem 3.4. The low rank approximation of a matrix in the 2-Norm can be minimized using the following theorem.

Theorem 3.4 (Schmidt, Mirsky, Eckard, Young)**.** *Given a matrix $\boldsymbol{A} \in \mathbb{C}^{n\times m}$ with rank$(\boldsymbol{A}) = r$. Let $\boldsymbol{USV}^* = \boldsymbol{A}$ denote the singular value decomposition of $\boldsymbol{A}$. Then for any $k < r$*

$$\min_{rank(X)=k} \|A - X\|_2 = \sigma_{k+1},$$

where $\{\sigma_i\}$ denote the descending singular values. Moreover, a non-unique minimizer is $X = \sum_{i=1}^{k} \sigma_i u_i v_i^*$, where $\{u_i\}$ and $\{v_i\}$ are the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ respectively.

A proof is given in [3]. The result also extends to the Frobenius norm, which is used for error bounds. In particular, this implies that the POD basis is the best approximation of the snapshot set $\mathcal{Y}^{\text{SNAP}}$ provided a given rank ℓ.

Note that the corresponding Hilbert space $\mathbb{R}^N$ does not reflect the properties of the original Hilbert space $L^2([0,1])$. It is reasonable to introduce a weighted inner product $\langle u, v\rangle_{\mathbf{M}} = u^\top \mathbf{M} v$ with the symmetric, positive definite mass matrix $\mathbf{M}$. The mass matrix naturally is

given by the FEM procedure. Given any functions $v(x)$ and $w(x)$ that are spanned by the Finite Element Basis, that is

$$v(x) = \sum_{i=1}^{N} \nu_i \phi_i(x) \qquad \text{and} \qquad w(x) = \sum_{i=1}^{N} \eta_i \phi_i(x),$$

the L^2 inner product is computed as

$$\begin{aligned} \langle v, w \rangle_{L^2([0,1])} &= \int_0^1 v(x)\overline{w(x)}dx \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \nu_i \eta_j \int_0^1 \phi_i(x)\phi_j(x)dx \\ &= \tilde{v}^\top \mathbf{M} \tilde{w} = \langle \tilde{v}, \tilde{w} \rangle_{\mathbf{M}}. \end{aligned} \tag{3.7}$$

Here, $\tilde{v}$ and $\tilde{w}$ denote the vectors of the nodal coefficients of $v(x), w(x)$ respectively, that is

$$\tilde{v} = \begin{pmatrix} \nu_1 \\ \nu_2 \\ \vdots \\ \nu_N \end{pmatrix} \qquad \text{and} \qquad \tilde{w} = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_N \end{pmatrix}.$$

In other words, the L^2 inner product is given by the weighted inner product of the nodal coefficient vectors with the mass matrix M. Using the weighted inner product in in (3.6) yields the generalized eigenvalue problem (for details see [68])

$$\left(\mathbf{M}\mathbf{Y}^{\text{SNAP}}\right)\left(\mathbf{M}\mathbf{Y}^{\text{SNAP}}\right)^\top \phi = \lambda \mathbf{M}\phi. \tag{3.8}$$

The weighted inner product is necessary to force the projection matrix $\mathbf{V}_\ell$ to be $\mathbf{M}$- orthonormal. The eigenvalue problem (3.8) can be solved by using a singular value decomposition as outlined in Algorithm 2, which for historical reasons is known as *Method of Snapshots* [64]. Using the representation (3.7) of the L^2 inner product we compute:

$$\begin{aligned} \langle \phi_i^P, \phi_j^P \rangle_{L^2([0,1]} &= \langle \phi_i^P, \phi_j^P \rangle_{\mathbf{M}} \\ &= \left(\phi_i^P\right)^T \mathbf{M} \phi_j^P \\ &= \left(\mathbf{L}^T \phi_i^P\right)^T \left(\mathbf{L}^T \phi_j^P\right) \\ &= \left(\mathbf{L}^T \mathbf{L}^{-T} u_i\right)^T \left(\mathbf{L}^T \mathbf{L}^{-T} u_j\right) \\ &= u_i^T u_j = \delta_{ij} \end{aligned}$$

and the POD basis is orthonormal as desired. Here, u_i $(i = 1, \ldots, N)$ denotes the columns of the matrix $\mathbf{U}$ of the singular value decomposition as described in Algorithm 2. The projection matrix $\mathbf{V}_\ell$ is determined by the first ℓ POD basis vectors, that is

$$\mathbf{V}_\ell = \begin{pmatrix} \phi_1^P & \phi_2^P & \cdots \phi_\ell^P \end{pmatrix} \in \mathbb{R}^{N \times \ell}.$$

Algorithm 2 Method of Snapshots

Input: Snapshot matrix $\mathbf{Y}^{\text{SNAP}} \in \mathbb{R}^{N\times M}$, mass matrix $\mathbf{M} \in \mathbb{R}^{N\times N}$
Output: POD basis $\{\phi_i^P\}$.

1. Compute the Cholesky decomposition of $\mathbf{M}$

$$\mathbf{L}^\top \mathbf{L} = \mathbf{M}$$

2. Compute the singular value decomposition of $\mathbf{W} = \mathbf{L}^\top \mathbf{Y}^{\text{SNAP}}$

$$\mathbf{U}\mathbf{S}\mathbf{V}^\top = \mathbf{W}$$

3. Set

$$\mathbf{POD} = \mathbf{L}^{-\top}\mathbf{U}.$$

ℓ	Finite Differences singular value	Finite Differences $\mathcal{E}(\ell)$	Finite Element Method singular value	Finite Element Method $\mathcal{E}(\ell)$	Group Finite Elements singular value	Group Finite Elements $\mathcal{E}(\ell)$
1	1.000000	0.788443	1.000000	0.789885	1.000000	0.789135
2	0.193533	0.941033	0.192597	0.942015	0.193426	0.941774
3	0.047495	0.978480	0.046976	0.979120	0.047268	0.979075
4	0.015563	0.990751	0.015250	0.991166	0.015336	0.991178
5	0.006225	0.995659	0.006026	0.995927	0.006045	0.995948
6	0.002817	0.997880	0.002689	0.998051	0.002688	0.998070
7	0.001351	0.998946	0.001268	0.999052	0.001263	0.999066
8	0.000662	0.999468	0.000608	0.999533	0.000603	0.999542

Table 3.1: Singular values for the different discretization techniques (Example 2, Euler)

A crucial question is the choice of ℓ. It appears that there is no a-priori rule on how to choose the dimension ℓ of the low-rank approximation [68]. A common heuristic choice is based on the *energy percentage*, which is defined as the ratio of the energy contained in the reduced model and the energy contained in the full model, that is

$$\mathcal{E}(\ell) = \frac{\sum_{i=1}^{\ell} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2}$$

where $\{\sigma_i\}_{i=1}^n$ denote the singular values. This criteria might fail because it is possible that dynamically important POD modes of lower energy are omitted [61]. In Figure 3.1, the decay of the normalized singular values for the different discretization techniques is presented. We used Example 1 (compare Section 2.5) with $N = 48$ for the computation, and an implementation of the backward Euler with $M = 80$ time steps as ODE solver. As outlined in Table 3.1, the singular values and the energy percentage do not differ much between the discretization techniques. The energy percentage of the GFE is slightly better then of the other methods. The heuristic should lead to better results, if the POD basis

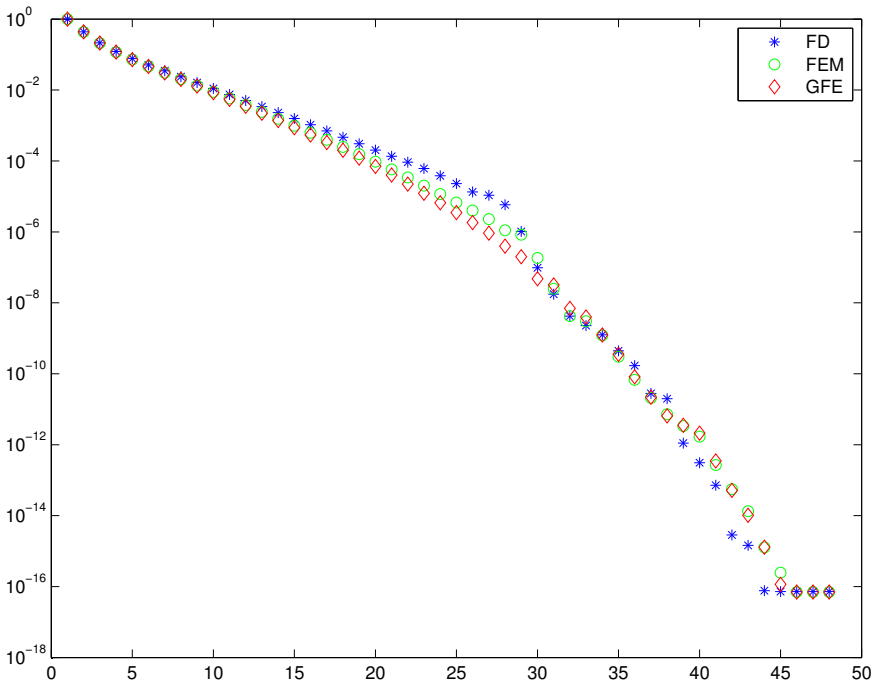


Figure 3.1: Decay of the normalized singular values (Example 2, Euler)

constructed with GFE is applied. Figure 3.2 suggests that there is hardly any difference within the POD basis functions.

3.1.2 POD Basis in L^2

A natural extension, is the derivation of the POD basis in the Hilbert space $L^2(\Omega)$. We assume that $\Omega \subseteq \mathbb{R}^N$ for some $N \in \mathbb{N}$. Given the dynamical system

$$\begin{cases} \dfrac{\partial}{\partial t} y(t,x) = f(t,x,y(t,x)) & x \in \Omega, t \in (0,T), \\ y(0,x) = y_0(x) & x \in \Omega, \end{cases} \tag{3.9}$$

we want to compute the POD basis of rank ℓ, that is find an orthonormal set of functions $\{\phi_i : \Omega \to \mathbb{R}\}_{i=1}^{\ell} \subseteq L^2(\Omega)$ that approximates the trajectory $\{y(t,\cdot) : t \in [0,T)\}$ of the solution of (3.9) best. This yields the optimization problem

$$\min_{\{\phi_i\} \in L^2(\Omega)} \int_0^T \left(\int_\Omega \left| y(t,x) - \sum_{i=1}^{\ell} \left(\int_\Omega y(t,z)\phi_i(z)dz \right) \phi_i(x) \right|^2 dx \right) dt \tag{3.10}$$

subject to $\langle \phi_i, \phi_j \rangle_{L^2(\Omega)} = \delta_{ij}$. We first focus on the case $\ell = 1$. Then (3.10) becomes

$$\min_{\phi \in L^2(\Omega)} \int_0^T \left(\int_\Omega \left| y(t,x) - \langle y(t,\cdot), \phi \rangle_{L^2(\Omega)} \phi(x) \right|^2 dx \right) dt \qquad \text{s.t.} \qquad \|\phi\|^2_{L^2(\Omega)} = 1. \tag{3.11}$$

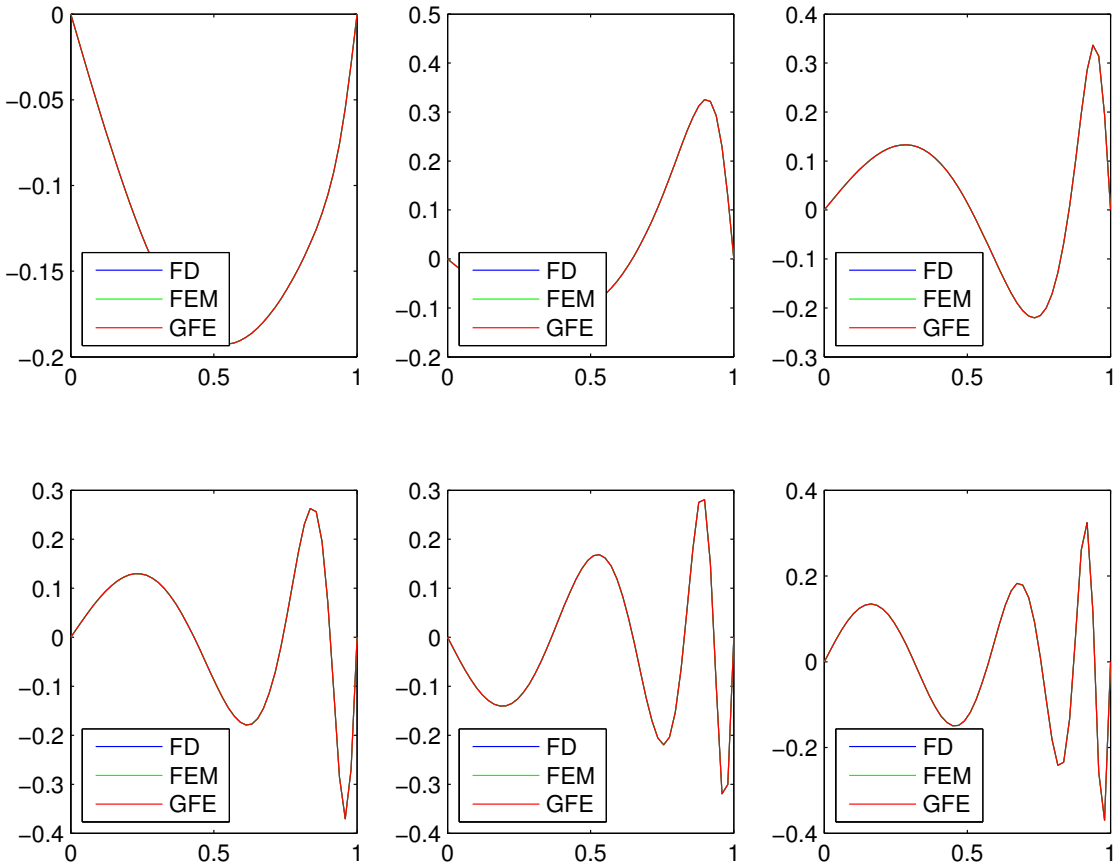


Figure 3.2: Example 1: First six POD modes

Since $L^2(\Omega)$ is a separable Hilbert space, there is a set of functions $\{\phi_i\}_{i=2}^{\infty} \subseteq L^2(\Omega)$ such that $\{\phi\} \cup \{\phi_i\}_{i=2}^{\infty}$ is an orthonormal basis in $L^2(\Omega)$. Hence, we express y as

$$y(t,x) = \langle y(t,\cdot),\phi\rangle_{L^2(\Omega)}\phi(x) + \sum_{i=2}^{\infty}\langle y(t,\cdot),\phi_i\rangle_{L^2(\Omega)}\phi_i(x).$$

Using this representation, Pythagoras' theorem and the fact that one can exchange an infinite sum and an integral if the sum is absolute convergent, we conclude

$$\begin{aligned}
&\int_0^T \left(\int_\Omega \left|y(t,x) - \langle y(t,\cdot),\phi\rangle_{L^2(\Omega)}\phi(x)\right|^2 dx\right) dt \\
= &\int_0^T \|y(t,\cdot\cdot) - \langle y(t,\cdot),\phi\rangle_{L^2(\Omega)}\phi\|^2_{L^2(\Omega)} dt = \int_0^T \left\|\sum_{i=2}^{\infty}\langle y(t,\cdot),\phi_i\rangle_{L^2(\Omega)}\phi_i(x)\right\|^2_{L^2(\Omega)} dt \\
= &\int_0^T \left(\sum_{i=2}^{\infty}\|\langle y(t,\cdot),\phi_i\rangle_{L^2(\Omega)}\phi_i\|^2_{L^2(\Omega)}\right) dt = \sum_{i=2}^{\infty}\int_0^T \left|\langle y(t,\cdot),\phi_i\rangle_{L^2(\Omega)}\right|^2 dt.
\end{aligned}$$

Following Volkwein [68], we apply the Lagrange-Framework and introduce the Lagrange-Functional $\mathcal{L} : L^2(\Omega)\times\mathbb{R}\to\mathbb{R}$ by

$$\mathcal{L}(\phi,\lambda) = \int_0^T \left(\int_\Omega |y(t,x)\phi(x)|^2 dx\right) dt + \lambda\left(1 - \int_\Omega |\phi(x)|^2 dx\right).$$

A first-order necessary optimality-condition is given by $\nabla\mathcal{L}(\phi,\lambda) \stackrel{!}{=} 0$ in $L^2(\Omega)\times\mathbb{R}$. We first compute the partial derivative of $\mathcal{L}$ with respect to ϕ:

$$\nabla_\phi\mathcal{L}(\phi,\lambda) = 2\int_0^T\int_\Omega y(t,x)\phi(x)dxy(t,z)dt - 2\lambda u(z) \stackrel{!}{=} 0. \tag{3.12}$$

The partial derivative with respect to λ is given by $\nabla_\lambda\mathcal{L}(\phi,\lambda) = 1 - \int_\Omega|\phi(x)|^2dx$, which yields the side constraint

$$\|\phi\|^2_{L^2(\Omega)} = 1.$$

Assuming (3.12) is well-defined, we use Fubini's theorem to receive a Fredholm integral equation of the form

$$\int_\Omega\left(\underbrace{\int_0^T y(t,x)y(t,z)dt}_{:=R(x,z)}\right)\phi(x)dx = \lambda\phi(z). \tag{3.13}$$

Define the operator $\mathcal{R}\phi = \int_\Omega R(x,z)\phi(z)dx$, which is a compact self-adjoint operator if the conditions specified in [34] hold. In this case, Hilbert-Schmidt theory guarantees that we can solve (3.13) with a countable set of decaying eigenvalues and corresponding eigenfunctions. The eigenfunction corresponding to the largest eigenvalue solves the maximization problem (3.11). Moreover, if $\{\phi_i\}_{i=1}^\ell$ solves (3.13) with descending corresponding eigenvalues, then $\{\phi\}_{i=1}^\ell$ solves the optimization problem (3.10). Therefore we have the following. The POD basis $\{\phi_i\}_{i=1}^\ell$ of (3.9) is given by the solution of the eigenvalue problem

$$\mathcal{R}\phi_i = \lambda\phi_i \qquad \text{in } \Omega.$$

This is equivalent to

$$\int_\Omega \frac{1}{T}R(x,z)\phi_i(x)dx = \lambda\phi(z) \qquad z\in\Omega,$$

where the factor $\frac{1}{T}$ is absorbed in λ. We conclude with a definition from [34].

Definition 3.5 (Autocorrelation function)**.** *The function $R(x,z)$ is called* empirical autocorrelation function.

Some authors, for example Jeff Borggaard and Traian Iliescu [70], include the factor $\frac{1}{T}$ in their definition of the autocorrelation function.

3.1.3 Structure of the POD Basis

The structure of the POD basis is analyzed in this section. All computations of this section are based on the FD approximation combined with the backward Euler with the parameter setup presented in Table 3.2. Recall the approximate solution for Example 1 as presented in Figure 2.2. The first POD mode (see Figure 3.2) coincides with the time average of the approximate solution. The time average is given by

$$\overline{y}(x) := \frac{1}{T}\int_0^T y(t,x)dt,$$

N	M	T	ν
48	80	1	0.01

Table 3.2: Parameter setup for analyzing the POD basis

where y denotes the solution of Burgers' equation. In the discretized case, the time average is given by

$$\overline{y}_i := \frac{1}{M}\sum_{j=1}^{M} y_i(t_j) \qquad \text{for} \qquad i = 0, \ldots, N+1.$$

The time average approximates the first POD mode very well as one can verify in Figure 3.3. Note that the negative first POD mode almost coincides with the time average. Since the POD basis is an orthonormal basis, multiplication with -1 does not change its properties. An explanation is given in the following remark. Since the first singular value contains almost

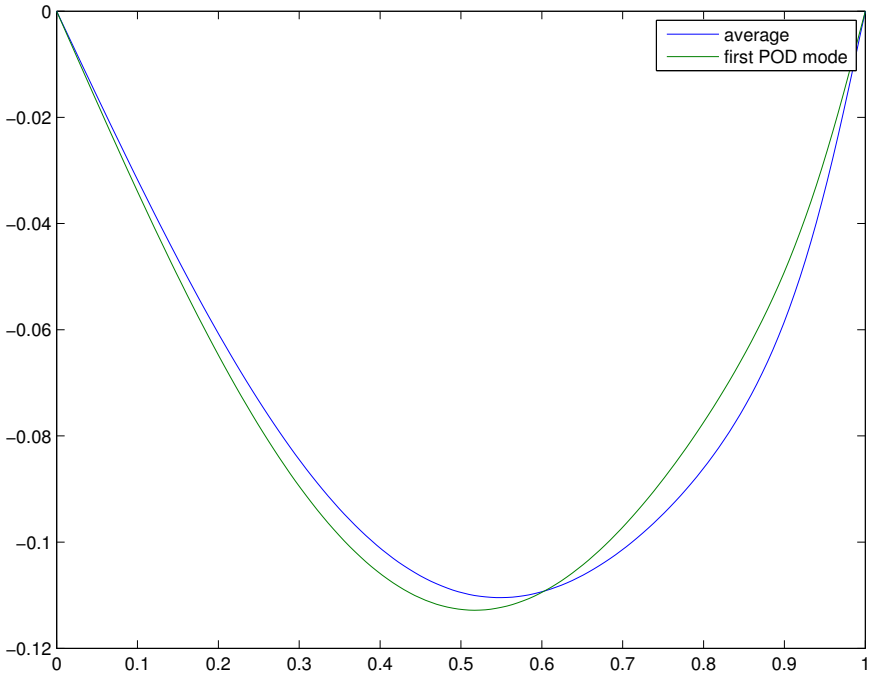


Figure 3.3: First POD mode compared with the time average

80% of the energy (compare Table 3.1), we assume that the solution consists basically of the first POD mode. Let y be a solution of Burgers' equation with decomposition

$$y(t,x) = \overline{y}(x) + \varepsilon\hat{y}(t,x) \tag{3.14}$$

in the time average with small remaining term $\varepsilon\hat{y}$. As outlined in the Subsection 3.1.2, computing the analytical POD basis results in the eigenvalue problem

$$\frac{1}{T}\int_0^T \int_0^1 y(t,x)y(t,z)\phi(z)dz = \lambda\phi(x). \tag{3.15}$$

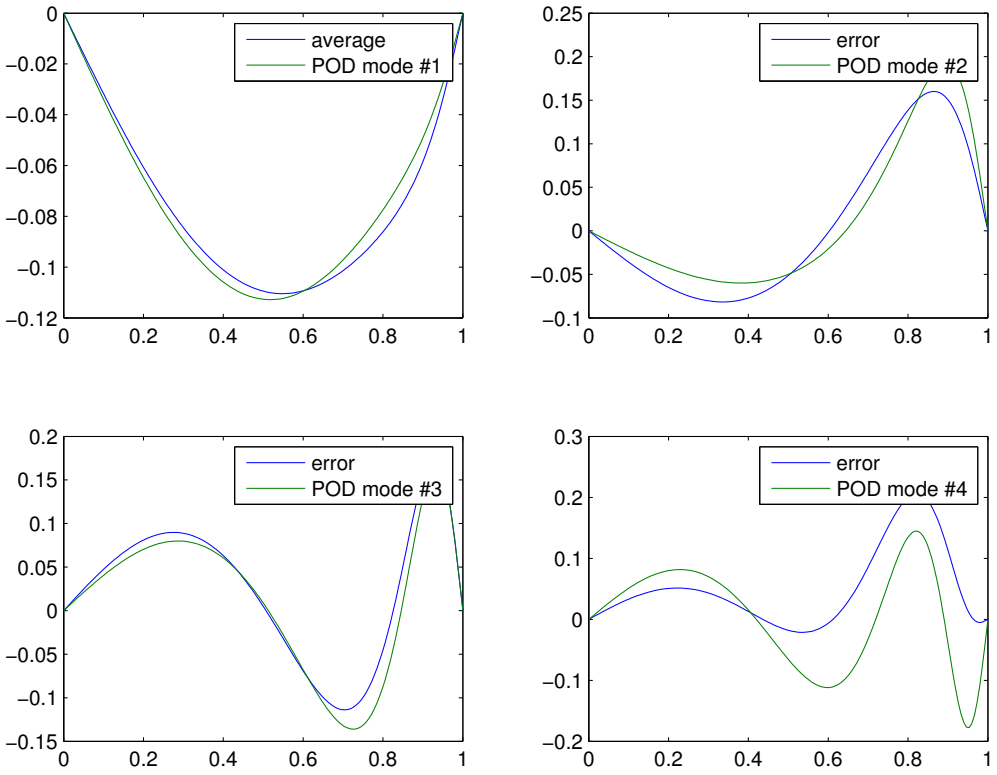


Figure 3.4: Example 1: First four POD modes compared with the time average and resulting errors

Substituting (3.14) in (3.15) yields

$$\frac{1}{T}\int_0^T \int_0^1 \overline{y}(x)\overline{y}(z)\phi(z)dz + \mathcal{O}(\varepsilon) = \lambda\phi(x).$$

Calculating

$$\frac{1}{T}\int_0^T \int_0^1 \overline{y}(x)\overline{y}(z)\phi(z)dz = \overline{y}(x)\underbrace{\int_0^1 \overline{y}(z)\phi(z)dz}_{=:c} = c\overline{y}(x)$$

shows that the eigenfunction is approximately given by the (normalized) time average. We obtain a similar result if we compute the error between the time average and the first POD mode, normalize it and compare it to the second POD mode. This scheme continues (Figure 3.4). We can use this observation to allocate a new algorithm for the determination of the POD basis. It is inexpensive to compute the time average and a power iteration of the matrix $\left(\mathbf{Y}^{\text{SNAP}}\right)^T \mathbf{Y}^{\text{SNAP}}$ calculates the first POD mode. Fast convergence is achieved since our initial vector consists almost only of the eigenfunction corresponding to the biggest eigenvalue. The errors and the inverse power iteration with a shifting strategy are employed to determine the next POD modes. Algorithm 3 summarizes the strategy. This works as far as the difference between the eigenvalues is large enough. However, it might fail for close eigenvalues. This is the case in the present situation, since the singular values decay close to zero. Beside that, the inverse power iteration is not cheaper than a singular value decomposition and hence the outlined method has no computational advantage. Moreover, the

Algorithm 3 Alternative POD basis algorithm

Input: Snapshot matrix $\mathbf{Y}^{\text{SNAP}} \in \mathbb{R}^{N\times M}$ **Output:** POD basis $\{\phi_i^P\}$.

1. Compute the time average $\overline{y}$ of the snapshot matrix $\mathbf{Y}^{\text{SNAP}}$.
2. Find the first POD mode via power iteration and use $\overline{y}$ to initialize the power iteration.
3. Compute the normalized error ERR between the POD mode and the time average.
4. For $i = 2, \ldots, \ell$:
   - Set
   $$\lambda = \frac{\text{ERR}^\top \mathbf{Y}^{\text{SNAP}} \left(\mathbf{Y}^{\text{SNAP}}\right)^\top \text{ERR}}{\|\text{ERR}\|^2}$$
   as initial shift.
   - Apply the inverse power iteration with initial vector ERR and shift λ.
   - Compute the new error as the normalized error between the POD mode and the previous error.

proposed algorithm is neither robust nor stable and does not work in general. In Example 2, the singular values are not decaying as fast as in Example 1, and indeed, the results from the procedure outlined above give results not as good as in Example 2 (Figure 3.5). The second example leads to another point in the discussion of the structure of the POD modes. In Figure 3.6 we present the first POD modes of Example 2. We recover a trigonometric behavior of the POD modes in the interval $[0.5, 1]$. The higher the POD mode the stronger the alternation of the function. Recall the solution plot of Example 2 presented in Figure 2.3 to understand this behavior.

Within the interval $[0.5, 1]$, the solution behaves like a single wave traveling through the domain. To model this behavior, consider the *one-way wave equation*

$$\frac{\partial y}{\partial t}(x,t) = a\frac{\partial y}{\partial x}(x,t) \qquad \text{in } \mathbb{R}\times(t_1,t_2), \tag{3.16a}$$

with initial condition

$$y(x,0) = f(x) \qquad \text{in } \mathbb{R}. \tag{3.16b}$$

Here, $a \in \mathbb{R}$ is constant, and $f : \mathbb{R} \to \mathbb{R}$ is a function. One-way wave-equations are widely used in underwater acoustics and geophysics [30] and as energy-absorbing boundary conditions [25]. For a first observation we assume $f \in C^1(\mathbb{R})$. One can easily check that

$$y(x,t) = f(x+at) \tag{3.17}$$

is a solution of (3.16). In particular, the solution is a translation in time (see Figure 3.7).

Theorem 3.6 (POD basis for the one-way wave equation)**.** *The POD basis for the one-way wave equation* (3.16) *is given by the common Fourier basis, that is*

$$\phi_k(x) = e^{2\pi i k x} \qquad \textit{for} \qquad k \in \mathbb{N},$$

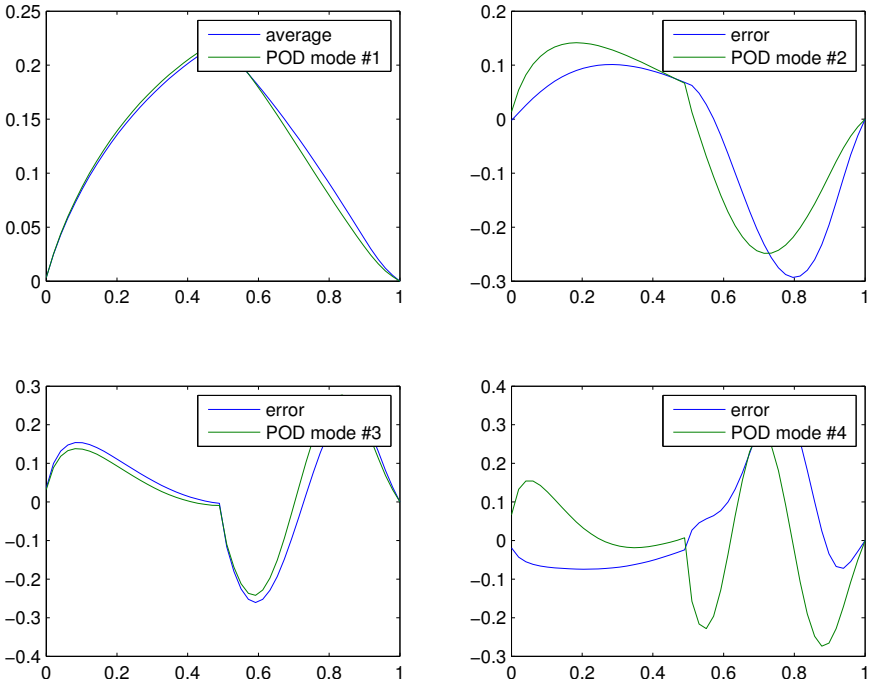


Figure 3.5: Example 2: First four POD modes compared with the time average and resulting errors

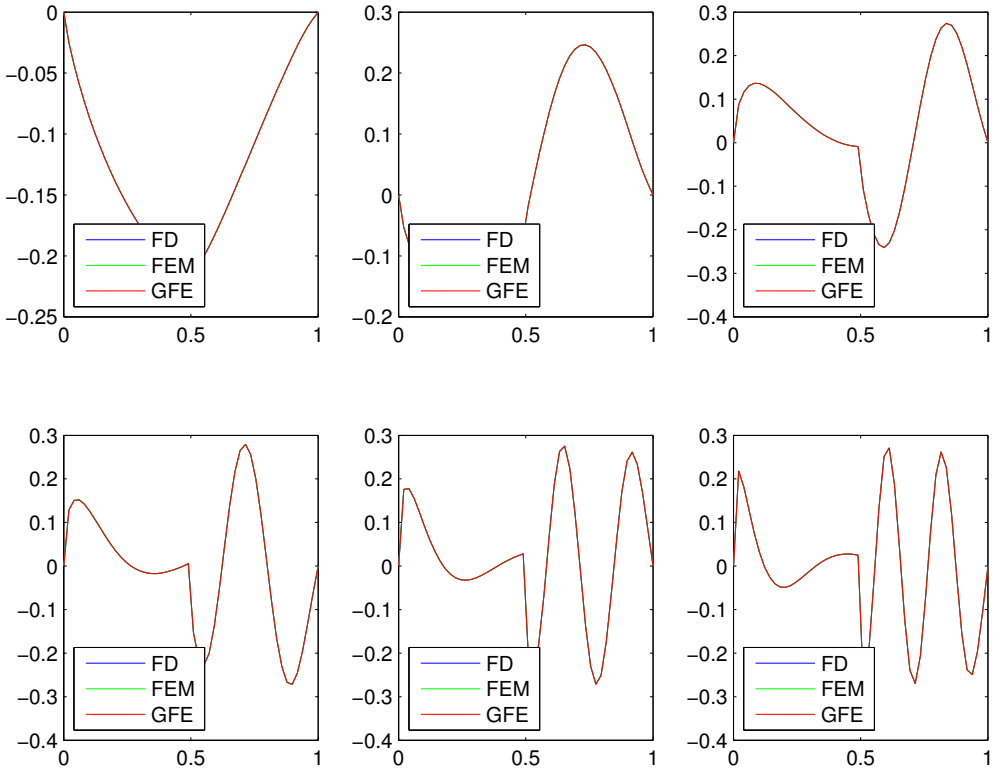


Figure 3.6: Example 2: First six POD modes

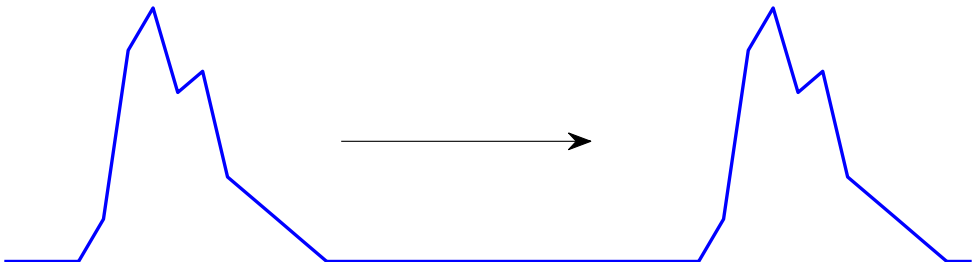


Figure 3.7: Solution of the one-way wave equation: A translation in time

where i denotes the imaginary unit.

Proof. According to the derivation in Subsection 3.1.2, the POD basis $\{\phi_k(\cdot)\}$ solves the eigenvalue problem

$$\int_{-\infty}^{\infty} R(x,z)\phi_k(x)dx = \lambda\phi_k(z) \qquad z \in \Omega. \tag{3.18}$$

Recall that the solution of the wave equation is given by (3.17). Hence, the autocorrelation function R is

$$R(x,z) = \int_{-\infty}^{\infty} f(x+at)f(z+at)dt \qquad \text{for} \qquad x,y \in \Omega.$$

The transformation $s = x + at$ yields

$$R(x,z) \;=\; \int_{-\infty}^{\infty} f(x+at)f(z+at)dt = \int_{-\infty}^{\infty} \frac{1}{a} f(s)f(s+(z-x))ds =: \mathcal{R}(z-x).$$

We deduce that $R(x,z) = \mathcal{R}(z-x)$ only depends on the distance $z-x$. As outlined in [34], this property is called *homogeneous* in turbulence literature. In particular, $R(x,z)$ has a Fourier representation of the form

$$R(x,z) = \mathcal{R}(z-x) = \sum_{k=-\infty}^{\infty} c_k e^{2\pi ik(z-x)} = \sum_{k=-\infty}^{\infty} c_k e^{2\pi ikz} e^{-2\pi ikx}. \tag{3.19}$$

Using this representation, (3.18) becomes

$$\int_{t_1}^{t_2} \sum_{k=-\infty}^{\infty} c_k e^{2\pi ikz} e^{-2\pi ikx} \phi(x)dx = \lambda\varphi(z).$$

We observe that the eigenfunctions are given by $\phi_k(x) = e^{2\pi ikx}$, which is the Fourier basis. □

The POD basis contains no specific information of the solution of (3.16), since it is given by the Fourier basis. This is not a surprising effect since POD tries to extract the most important

information over time. Recalling that the solution is a translation in time, a bump traveling in time is comprehensible. Hence, there is no specific information at one point over time. The above observation coincides with [12]. We draw the following conclusion for the original problem. In Example 2 we cannot expect a good approximation with few basis modes. The Fourier basis contains no information of the specific solution and the POD basis in the interval $[0.5, 1]$ is very similar to the Fourier basis.

3.2 ROM with POD

The goal of this section is the derivation of the ROM for Burgers' equation with the POD basis from Subsection 3.1.1. We employ a standard Galerkin projection for the derivation, which results in a reduced system of the form

$$\dot{\alpha}_r(t) = \mathbf{A}_r \alpha_r(t) + \mathbf{V}_\ell^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) + \mathbf{B}_r \overline{f}(t). \tag{3.20}$$

In case of FEM the reduced model can directly be derived from the weak form (2.7) introduced in Chapter 2. For $k = 1, \ldots, \ell$ we have

$$\phi_k^P(x_i) = \sum_{j=1}^{N} \gamma_{kj} \phi_j(x_i),$$

where $\gamma_{kj} = \phi_k^P(x_j)$ are the nodal coefficients. In particular, it is

$$\left(\mathbf{V}_\ell\right)_{kj} = \gamma_{kj}.$$

Hence, the function $\sum_{j=1}^{N} \gamma_{kj}\phi_j(x)$ interpolates the POD mode $\phi_k^P(x)$ for $k = 1, \ldots, \ell$. Some researchers, for example Jarvis [38], say that the POD modes *live* in the FEM space. Functions in the FEM space are linear splines, while in case of the one-way wave equation the POD modes have no representation as linear splines. In contrast, the discretized POD basis consists of linear splines and therefore the POD mode has a FE extension.

For the derivation of the reduced model, substitute the POD approximation

$$y(t,x) \approx y^P(t,x) = \sum_{i=1}^{\ell} \alpha_i^P(t)\phi_i^P(x) = \sum_{i=1}^{\ell} \alpha_i^P(t) \sum_{j=1}^{N} \gamma_{ij}\phi_j(x) \tag{3.21}$$

in the weak form (2.7) and compute the resulting matrices term-by-term. Since the procedure is the same as in the FE case, we only present the computations for the stiffness matrix at this point.

$$\begin{aligned}
\langle \phi_{i,x}^P, \phi_{j,x}^P \rangle_{L^2([0,1])} &= \int_0^1 \phi_{i,x}^P(x)\phi_{j,x}^P(x)dx = \int_0^1 \sum_{k=1}^{N} \gamma_{ik}\phi_{k,x}(x) \sum_{l=1}^{N} \gamma_{jl}\phi_{l,x}(x)dx \\
&= \sum_{k=1}^{N}\sum_{l=1}^{N} \gamma_{ik}\gamma_{jl} \underbrace{\int_0^1 \phi_{k,x}(x)\phi_{l,x}(x)dx}_{=:a_{kl}}.
\end{aligned}$$

The reduced matrix $\mathbf{A}_r$ can be written in terms of the POD coefficients, and its decomposition is given as

$$\mathbf{A}_r = \mathbf{V}_\ell^\top \mathbf{A} \mathbf{V}_\ell \in \mathbb{R}^{\ell \times \ell},$$

which coincides with the former representation introduced in Section 3.1. The same procedure yields the matrix $\mathbf{B}_r$ and shows that the mass matrix of the reduced system is given by the identity. It follows a discussion of the nonlinear term for the different discretization techniques.

Finite Differences The nonlinear term in the FD approximation is given by

$$\mathcal{N}_{\text{FD}}(\alpha(t)) = \alpha(t) \bullet \underbrace{\frac{1}{2h} \begin{pmatrix} 0 & -1 & & & \\ 1 & & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & & -1 \\ & & & 1 & 0 \end{pmatrix}}_{=:\mathbf{N}_{\text{FD}}} \alpha(t).$$

Replacing $\alpha(t)$ by $\mathbf{V}_\ell \alpha_r(t)$ yields the desired nonlinear term in the reduced-order model. Note that $\mathbf{N}_{\text{FD},r} = \mathbf{N}_{\text{FD}} \mathbf{V}_\ell \in \mathbb{R}^{N \times \ell}$ can be precomputed. The nonlinear term still depends on the original dimension N.

Finite Element Method In terms of FEM, the nonlinear term reads as

$$\mathcal{N}_{\text{FEM}}(\alpha(t)) = \frac{1}{6} \begin{pmatrix} -\alpha_1(t)\alpha_2(t) - \alpha_2^2(t) \\ \alpha_1^2(t) + \alpha_1(t)\alpha_2(t) - \alpha_2(t)\alpha_3(t) - \alpha_3^2(t) \\ \vdots \\ \alpha_{N-2}^2(t) + \alpha_{N-2}(t)\alpha_{N-1}(t) - \alpha_{N-1}(t)\alpha_N(t) - \alpha_N^2(t) \\ \alpha_{N-1}^2(t) + \alpha_{N-1}(t)\alpha_N(t) \end{pmatrix} \in \mathbb{R}^N.$$

Again, we replace $\alpha(t) = \mathbf{V}_l \alpha_r(t)$ to determine the reduced system. For a better speedup, a matrix tensor is computed based on the weak form. Substituting (3.21) into (2.7) gives

$$-\int_0^1 y^P(t,x) y_x^P(t,x) \phi_i^P(x) dx \qquad \text{for } i = 1, \dots, \ell$$

for the i-th component of the nonlinear term. Using representation (3.21), we obtain

$$\begin{aligned} & -\int_0^1 y^P(t,x) y_x^P(t,x) \phi_i^P(x) dx \\ = \quad & -\int_0^1 \left[\left(\sum_{k=1}^{\ell} \alpha_k^P(t) \sum_{n=1}^{N} \gamma_{kn} \phi_n(x) \right) \left(\sum_{m=1}^{\ell} \alpha_m^P(t) \sum_{r=1}^{N} \gamma_{mr} \phi_r'(x) \right) \sum_{j=1}^{N} \gamma_{ij} \phi_j(x) \right] dx \\ = \quad & -\sum_{k=1}^{\ell} \sum_{m=1}^{\ell} \alpha_k^P(t) \alpha_m^P(t) \sum_{n=1}^{N} \sum_{r=1}^{N} \gamma_{kn} \gamma_{mr} \sum_{j=1}^{N} \gamma_{ij} \int_0^1 \phi_n(x) \phi_r'(x) \phi_j(x) dx. \end{aligned} \tag{3.22}$$

Observe that the integral in (3.22) is the same as in the nonlinear term when FEM is applied. See Appendix A for a detailed computation. Define the matrix $\mathbf{T}_i = (t_{i,nr}) \in \mathbb{R}^{N\times N}$ where

$$t_{i,nr} = -\sum_{j=1}^{N} \gamma_{ij} \int_0^1 \phi_n(x)\phi_r'(x)\phi_j(x)dx.$$

Then the i-th entry of the nonlinear term assembles as

$$\left(\mathbf{V}_\ell^\top \mathcal{N}_{\text{FEM}}\left(\mathbf{V}_\ell \alpha^P(t)\right)\right)_i = \alpha^P(t)^\top \underbrace{\mathbf{V}_\ell^\top \mathbf{T}_i \mathbf{V}_\ell}_{=:\mathbf{N}^r_{\text{FEM},i}\in\mathbb{R}^{\ell\times\ell}} \alpha^P(t).$$

We can precompute $\mathbf{N}^r_{\text{FEM},i} \in \mathbb{R}^{\ell\times\ell}$. Let

$$\mathbb{N}^r_{\text{FEM}} = \begin{bmatrix} \mathbf{N}^r_{\text{FEM},1} \\ \mathbf{N}^r_{\text{FEM},2} \\ \vdots \\ \mathbf{N}^r_{\text{FEM},\ell} \end{bmatrix} \in \mathbb{R}^{\ell\times\ell\times\ell}$$

denote the tensor of the nonlinear term. The resulting tensor product for the nonlinear term is in $\mathcal{O}(\ell^3)$ and does not depend on the original dimension. Substituting $\alpha(t) = \mathbf{V}_\ell \alpha^P(t)$ in the original FEM model is in the $\mathcal{O}(N^2\ell)$.

Computing the integral by hand is relatively easy in the one-dimensional case. If the PDE affects more spatial dimensions one integrates over an higher dimensional element. However, this is basically the same procedure as in a common FE approach and can be implemented in the same way.

Group Finite Elements Recall that the GFE is based on an additional approximation of the grouped nonlinearity. This is possible due to the property

$$\phi_i(x_j) = \delta_{ij}$$

of the FE basis functions. This property does not hold for the POD modes and an approximation of the grouped term is not directly possible. However, one can extend the idea of interpolation to a POD reduced model, called Group Proper Orthogonal Decomposition. This method is discussed in Section 3.3.

Finally, the reduced system does not feature the sparsity gained from the discretization techniques employed in Chapter 2. However, the low-rank system gives a large numerical speedup (see numerical results in Section 3.6).

3.3 ROM with Group Proper Orthogonal Decomposition (GPOD)

As mentioned in the previous chapter, the aim is a group approximation for the nonlinear term of Burgers' equation similar to the Group Finite Element Method. This approach

was introduced in 2010 by Dickinson and Singler [23] and is based on the master thesis of Dickinson [22]. We strive for an approximation of the quadratic term of the form

$$y(t,x)^2 \approx \left[y^P(t,x)\right]^2 = \sum_{i=1}^{\ell} \mathcal{F}_i(\alpha_i^P(t))\phi_i^P(x),$$

with unknown functions $\mathcal{F}_i : \mathbb{R} \to \mathbb{R}$. The nonlinear part can be approximated as follows:

$$\langle y(t,\cdot)_x^2, \phi_i^P \rangle_{L^2([0,1])} \approx \mathbf{V}_\ell^\top \mathbf{N}_{\mathrm{GFE}} \left[\operatorname{diag}\left(\mathbf{V}_\ell \alpha^P(t)\right)\right] \mathbf{V}_\ell \alpha^P(t).$$

We note that $\mathbb{N}_{\mathrm{GFE},r} = \mathbf{V}_\ell^\top \mathbf{N}_{\mathrm{GFE}} \in \mathbb{R}^{\ell \times N}$ can be precomputed, which is an advantage with regard to the computational effort. The implementation in MATLAB® is easy, and according to Dickinson and Singler, one saves some operations compared to the the original computations for the POD approach. For more details on the interpolation used to compute the functions $\mathcal{F}_i$ and the computational advantages, the interested reader is referred to [22] and [23].

3.4 Discrete Empirical Interpolation Method (DEIM)

Analyzing the reduced-order system

$$\dot{\alpha}_r(t) = \mathbf{A}_r \alpha_r(t) + \mathbf{V}_\ell^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) + \mathbf{B}_r \overline{f}(t). \tag{3.23}$$

raised from POD in the previous sections, reveals that the nonlinear term still has to be evaluated in the original dimension N. Beside that, the low-dimensional solution has to be lifted to the full space and projected back. Both procedures are computationally expensive and dominate the computational effort if ℓ is small enough and therefore prevent a better speedup. The Discrete Empirical Interpolation Method (DEIM), introduced by Chaturantabut [17] in 2008, which is based on the Empirical Interpolation Method (EIM) [8], tries to resolve the issue. Let

$$\mathcal{N}_r(t) = \mathbf{V}_\ell^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right)$$

denote the nonlinear part of the POD based reduced model. For convenience we introduce $\eta(t) = \mathcal{N}(\mathbf{V}_\ell \alpha_r(t))$. The goal is to approximate η by projection on a suitable subspace $\mathbf{U} \in \mathbb{R}^{N \times k}$, that is

$$\eta(t) \approx \mathbf{U} c(t). \tag{3.24}$$

This yields $\mathcal{N}_r(t) \approx \mathbf{V}_\ell^\top \mathbf{U} c(t)$, where the matrix $V^\top U \in \mathbb{R}^{\ell \times k}$ can be precomputed. Notice that the nonlinear part does not depend on the original dimension N. If $\approx$ is replaced by $=$ in (3.24), then the system is overdetermined for $c(t)$. DEIM proposes to extract rows by multiplication with a truncated permutation matrix $\mathbf{P}$:

$$\mathbf{P}^\top \eta(t) = (\mathbf{P}^\top \mathbf{U}) c(t).$$

Using this approximation, the nonlinear term can be approximated by

$$\mathcal{N}(t) \approx \mathbf{V}_\ell^\top \mathbf{U}(\mathbf{P}^\top \mathbf{U})^{-1}\mathbf{P}^\top \mathcal{N}(\mathbf{V}_\ell \alpha_r(t)). \tag{3.25}$$

At this point, the nonlinear term still has to be evaluated in the full dimension. However, if $\mathcal{N}$ is a component-wise function, we have

$$\mathbf{P}^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) = \mathcal{N}\left(\mathbf{P}^\top \mathbf{V}_\ell \alpha_r(t)\right),$$

where $\mathbf{P}^\top \mathbf{V}_\ell \in \mathbb{R}^{k\times \ell}$ can be precomputed. In this case, the evaluation of the nonlinear term is completely independent of the full dimension N. Unfortunately, in case of Burgers equation and many other nonlinear PDEs, the nonlinear term arising from a FE discretization is not a component-wise function. However it is possible to push the truncated permutation inside the nonlinear term. We give a short introduction on how to proceed in this case. See the references mentioned in the introduction of this chapter for more details. Algorihtm 4 outlines the procedure to determine the matrices $\mathbf{U}$ and $\mathbf{P}$.

Algorithm 4 Discrete Empirical Interpolation Method

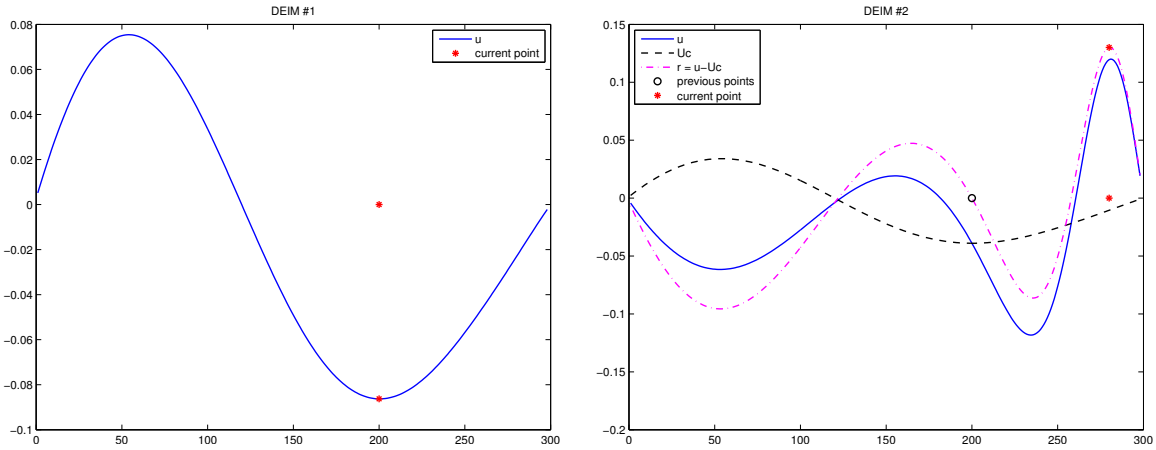
Input: $\mathbf{Y}^{\text{SNAP}}, k \in \mathbb{N}$

1. Evaluate the nonlinear term in $\mathbf{Y}^{\text{SNAP}}$, that is $\mathbf{N}^{\text{SNAP}} = \mathcal{N}\left(\mathbf{Y}^{\text{SNAP}}\right)$.
2. Compute the singular value decomposition of $\mathbf{N}^{\text{SNAP}}$: $\overline{\mathbf{U}}\mathbf{S}\mathbf{V}^\top = \mathbf{N}^{\text{SNAP}}$, and let $u_1, u_2, \cdots, u_k$ denote the first k columns of $\overline{\mathbf{U}}$.
3. Let $[\rho, p_1] = \max |u_1|$, $\mathbf{U} = [u_1]$, $\mathbf{P} = [e_{p_1}]$, where e_i denotes the i-th standard basis vector.
4. For $j = 2$ to m
   - $u \leftarrow u_j$.
   - Solve $(\mathbf{P}^\top \mathbf{U})c = \mathbf{P}^\top u$ for c.
   - $r = u - \mathbf{U}c$.
   - $[\rho, p_j] = \max |r|$.
   - $\mathbf{U} \leftarrow [\mathbf{U}\ u], \mathbf{P} = [\mathbf{P}\ e_{p_j}]$.

The first two iterations of Algorithm 4 are presented in Figure 3.8.

In our cases, the computation is more complicated since the nonlinear term is not a component-wise function. Assume we already have the matrices $\mathbf{P}, \mathbf{V}_\ell$ and $\mathbf{U}$ and try to simplify $\mathbf{P}^\top \mathcal{N}(\mathbf{V}_\ell \alpha_r(t))$. For each entry of the nonlinear function that is extracted by $\mathbf{P}$, determine the nonzero entries of $\mathbf{V}_\ell \alpha_r(t)$. Collect the indices of these entries and save them in a second truncated permutation matrix $\mathbf{P}_2 \in \mathbb{R}^{N\times k_2}$. The number $k_2 \in \mathbb{N}$ depends on the employed discretization technique. The nonlinear term is given by

$$\mathbf{P}^\top \mathcal{N}\left(\mathbf{V}_\ell \alpha_r(t)\right) = \mathbf{P}^\top \mathcal{N}\left(\tilde{\mathbf{V}}_\ell \alpha_r(t)\right),$$



(a) First iteration (b) Second iteration

Figure 3.8: Iterations of the DEIM algorithm

where $\tilde{\mathbf{V}}_\ell = \mathbf{P}_2^\top \mathbf{V}_\ell \in \mathbb{R}^{k_2 \times \ell}$ does not depend on the original dimension N. Due to the sparse structure of the nonlinear term, which is inherited from the discretization technique, $k_2 \ll N$ for $k \ll N$ holds, and we expect a numerical advantage.

Besides DEIM, there are other approaches to handle the costly nonlinear part. The reader is referred to the paper of Wang et al. [69] at this point.

3.5 Semi-implicit Euler

Since the derived reduced-order models shell still reflect the properties of the original system, we expect them to be stiff. In the original setting, we employed a common implicit Euler method beside the MATLAB® implementation for (stiff) ODE solvers. In General, there is no fast implementation of the gradient of the nonlinear term due to the projection matrix. And the explicit Euler does only work with a very fine time discretization. We resolve the issue with a method called *semi-implicit Euler*. This method is used in delay and stochastic equations [48]. Several stability and convergence results have been established, for example in [66]. Thanks to its symplectic property it can be used as a geometric numerical integrator [29]. The following review of the method is based on the work of Spigler and Vianello [66]. The authors provide convergence and consistency results. Consider a general evolution equation of the form

$$\dot{y} = \mathcal{F}(t, y(t)) + \mathcal{G}(t, y(t)).$$

Note that we can decompose the ROMs obtained from POD and DEIM in such a manner. To this end, we threat the linear and nonlinear term separately. Recall the definitions of the explicit and implicit Euler:

$$y_{n+1} = y_n + h\left[\mathcal{F}(t_n, y_n) + \mathcal{G}(t_n, y_n)\right] \qquad \text{(explicit Euler)} \qquad (3.26)$$

$$y_{n+1} = y_n + h\left[\mathcal{F}(t_{n+1}, y_{n+1}) + \mathcal{G}(t_{n+1}, y_{n+1})\right] \qquad \text{(implicit Euler)} \qquad (3.27)$$

The semi-implicit Euler combines the two approaches and has the formal iteration

$$y_{n+1} = y_n + h\left[\mathcal{F}(t_{n+1}, y_{n+1}) + \mathcal{G}(t_{n+1}, y_n)\right]. \tag{3.28}$$

In contrast to the explicit Euler (3.26), the time step t_{n+1} is still in place in (3.28). The system is implicit since the unknown y_{n+1} appears on both sides, but the choice of $\mathcal{F}$ and $\mathcal{G}$ is free. Recall that the nonlinearity in the implicit Euler yields a nonlinear system that one has to solve for example with Newton's method (Section 2.4). In contrast, the implicit Euler for a linear system is the solution of a linear system, which can be computed with a common linear solver. A reasonable choice for $\mathcal{F}$ and $\mathcal{G}$ in terms of the reduced system (3.4) is given by

$$\mathcal{F}(t, \alpha) = \mathbf{A}_r\alpha + \mathbf{B}_r\overline{f}(t) \qquad \text{and} \qquad \mathcal{G}(\alpha) = \mathbf{V}_\ell^\top \mathcal{N}\left(\mathbf{V}_\ell\alpha\right).$$

3.6 Numerical Results

We start the numerical result section with a numerical analysis of the alternative Algorithm 3 in Subsection 3.6.1. It follows a discussion of POD in Subsection 3.6.2 and we conclude with the analysis of DEIM in Subsection 3.6.3.

3.6.1 Alternative Algorithm for the POD Basis

We employ Algorithm 3 to determine its applicability to compute the POD basis in the alternative way proposed in Subsection 3.1.3. The parameter setup for the computations is the same as outlined in Table 3.2. First, let us examine the results for Example 1. In Figure 3.9, the first six alternative POD modes are presented. We can see that they are very similar. There also is a good agreement of the corresponding singular values of the first modes (see Table 3.3). The relative error of the POD modes is very small. But this does not hold for the latter modes. In accordance to this observation, the alternative POD modes do not match visually for $\ell = 7, 8$ and 9 as one can see in Figure 3.10 These results are gained by the following adjustment in Algorithm 3: Instead of computing the error of the POD mode and the previous error, the error between the POD mode and the vector after the first iteration of the power method is computed. This improves the accuracy and gives a better match in the singular values. For more details on the implementation, the interested reader is referred to the MATLAB® code provided in Appendix B. Besides POD mode number four, the accuracy for Example 2 is even better (Figure 3.11). Inspite of the above improvement of Algorithm 3, there are still mismatches in the POD modes and the obtained POD basis cannot be used since it is no longer an orthonormal basis. This coincides with the results of the discussion in Subsection 3.1.3: The proposed algorithm is not stable and fails in correctly determining the POD basis. Since the computation with Algorithm 3 takes longer than Algorithm 2, there is no reason to use the new method. But is is a results that helps to better understand the POD basis.

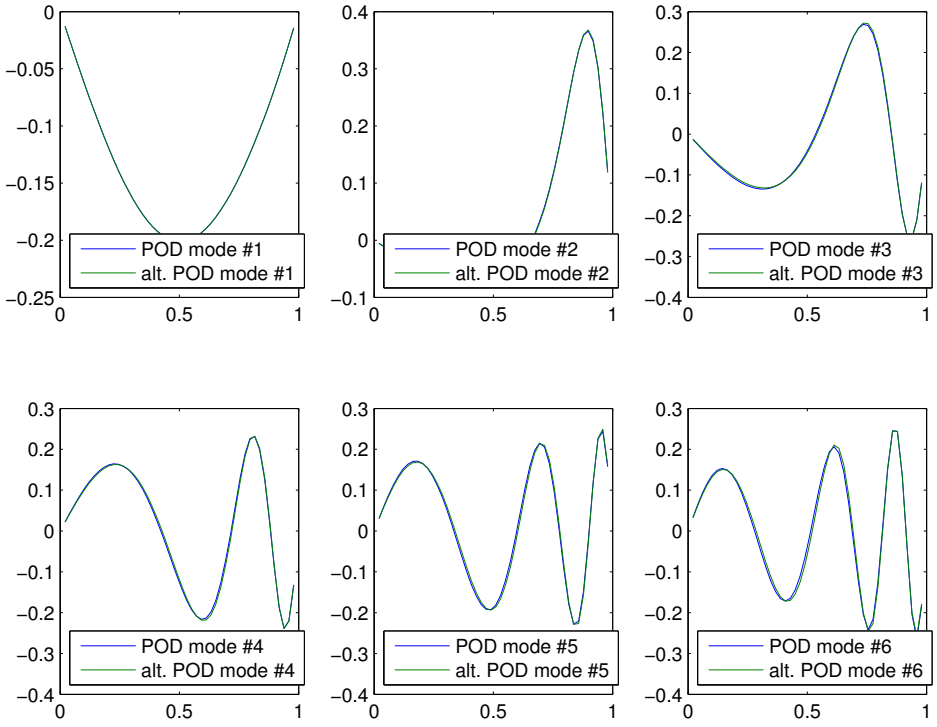


Figure 3.9: Example 1: Alternative POD modes

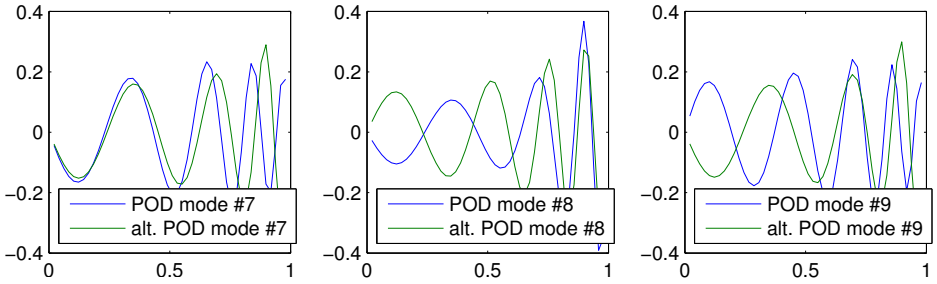


Figure 3.10: Example 1: Alternative POD modes for $\ell = 7, 8$ and 9

3.6.2 POD

In the following, POD (FD) means the ROM based on POD generated from the FD system. Accordingly, POD (FEM) is based on the FEM approximation. We present a comparison between the FEM approximation and the POD (FEM) approximation ($\ell = 2, N = 300, M = 300$) in Figure 3.12. The corresponding energy percentage is given by

$$\mathcal{E}(2) = 0.999632.$$

To that effect, the solution of the POD (FEM) reduced model approximates the FEM solution pretty well. Further POD modes have no effect on the approximation (compare Figure 3.13). We omit a further analysis of Example 1 due to less impact of further POD modes. The singular values and the engery percentage for Example 2 are presented in Table 3.1 in Subsection 3.1.1. The POD (FEM) approximation is presented in Figure 3.14 for $\ell = 5, 8, 11, 14$

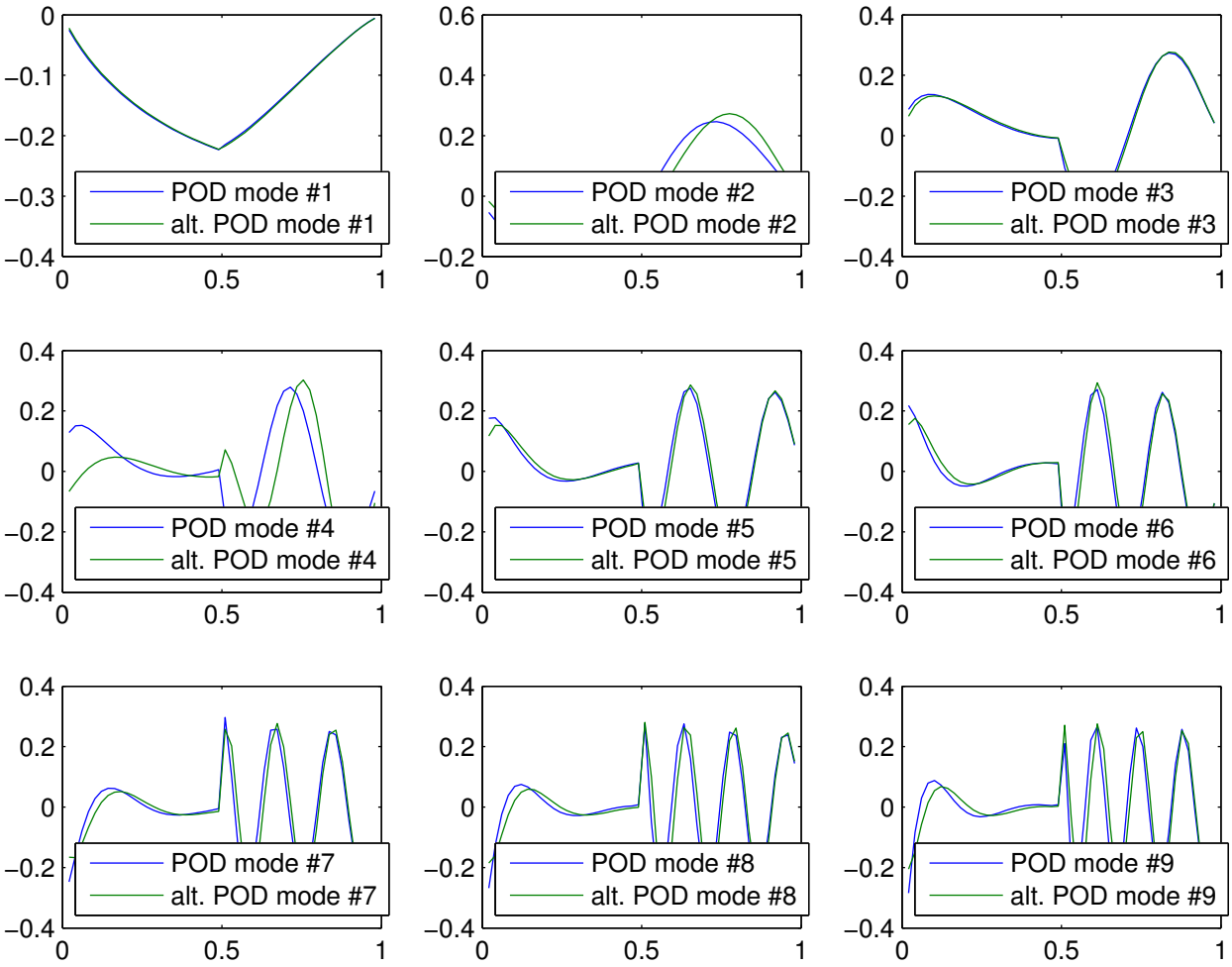
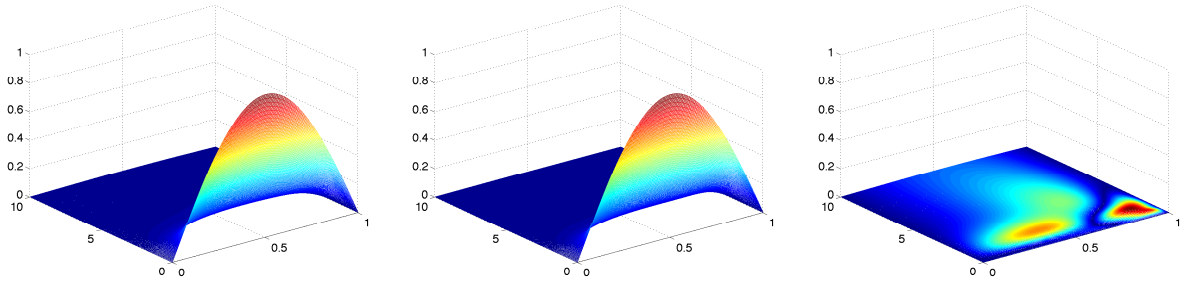


Figure 3.11: Example 2: Alternative POD modes

ℓ	POD	alternative POD	relative error
1	1.00000000	1.00000000	0.00039916
2	0.01089606	0.01081540	0.01125144
3	0.00167148	0.00160016	0.02774293
4	0.00028260	0.00026121	0.03912388
5	0.00003490	0.00003020	0.05332628
6	0.00000489	0.00000431	0.06747198
7	0.00000077	0.00000041	0.06747198
8	0.00000050	0.00000070	0.06747198
9	0.00000003	0.00000042	0.06747198

Table 3.3: Example 1: Normalized singular values for POD and the alternative POD modes



(a) FEM (b) POD (FEM) (c) Absolute error

Figure 3.12: Example 1: POD (FEM) approximation

and 17 POD modes. Figure 3.15 presents the corresponding absolute errors. Obviously, the error is larger in the interval $[0.5, 1]$ and the absolute error oscillates more. Moreover, one can recognize a strict border between the left and the right interval. This corresponds exactly to the result of Subsection 3.1.3. The L^2 errors of both intervals are presented in Table 3.4. The error in the left half interval is significantly smaller than the one in the right half interval. This holds for all numbers ℓ of employed POD modes. The overall (relative) error for Example 2 is ploted against the number of employed POD modes in Figure 3.16. A first observation is that the error obtained with the semi-implict Euler stays above 10^{-4}, while the error obtained by the other solver decreases further. This is based on the adaptive temporal grids the MATLAB® solvers employ in contrast to the equidistant grid of the semi-implicit Euler. For almost all ℓ the other solvers (ODE45, ODE15S and ODE23S) yield similar errors decaying to zero. For $\ell = 24$ POD modes, the error is less than 10^{-7}. This outlines the effectiveness of the POD even for nonsmooth initial conditions. To complete the discussion of the POD reduced models, we present time measurements for Example 2 in Table 3.5. The computations are based on $N = 100$ spatial and $M = 300$ temporal discretization points. The presented times are averaged over 50 iterations. The ODE23S solver yields the highest

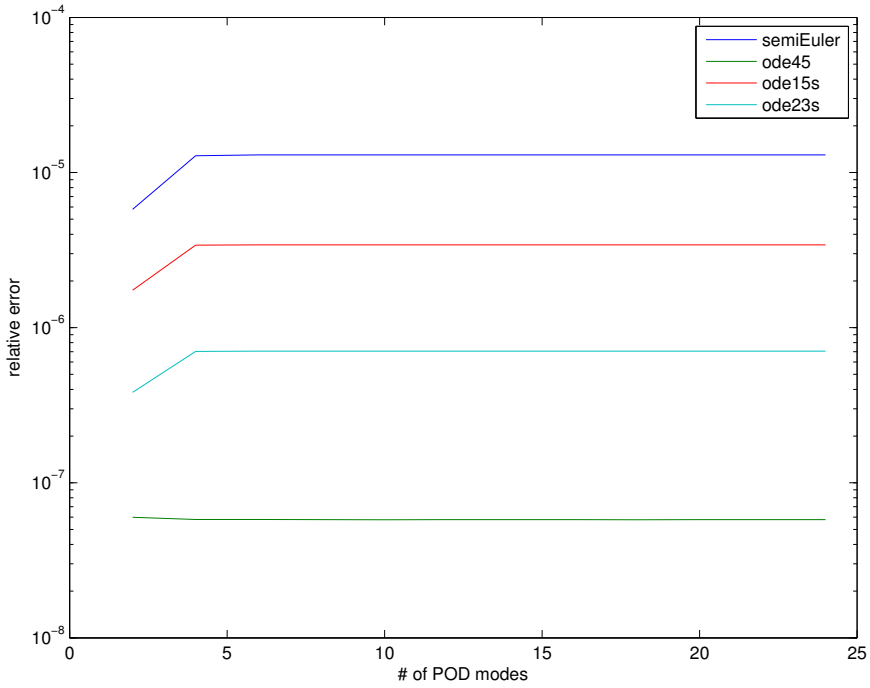
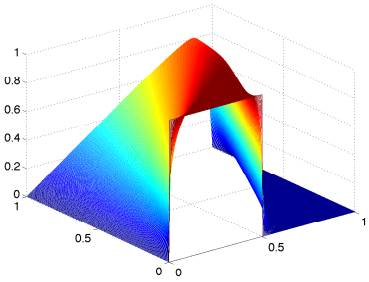
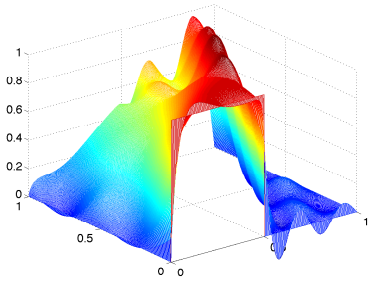


Figure 3.13: Example 1: POD (FD) relative error

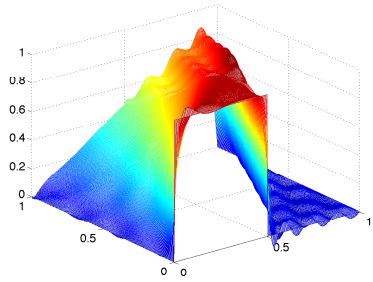
speedup for a low number of POD modes. As aspected, the computational time increases for more POD modes. The ODE15S takes more time for the POD reduced model ($\ell \geq 18$) than the full model. One the one side, the lift to the full dimension in the nonlinear term is an additional cost in the low rank model. On the other hand, ODE15 gets the best results for the full model. Its stiffness properties are obviously well suited for the orignial model but not for the reduced one. The analysis of the computational times confirms this assumption: All solver yield increasing times. For $\ell = 20$, the semi-implicit Euler with a fixed step size needs about 150% of the time with $\ell = 2$ POD modes. The adative solver yield much higher increases of the computational time. Hencefore, they require additional time steps in their computation to meet the integration tolerances. This implies that the POD reduced models get more stiff with additional basis functions. A more detailed analyis of the stiffness properties of POD reduced model is not yet considered, but future work.
Tor the semi-implict Euler, the ODE45 and the ODE23S solver, the POD model obtained from the FD discretized model yields faster times than POD (FEM) and GPOD. Only the ODE15S solver yields converse results. For the semi-implicit Euler, POD (FD) is significantly faster, the same holds for the ODE45 solver. This is based on the more efficient implementation of the nonlinear term. For high number of POD modes, the semi-implicit Euler yields the fastest times. However, as seen in Figure 3.16, it also provides a worse approximation of the full model.



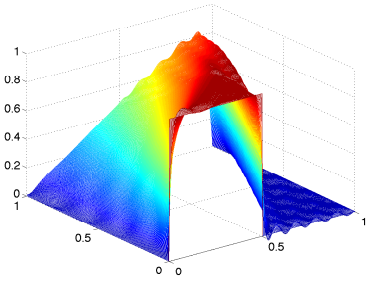
(a) FEM



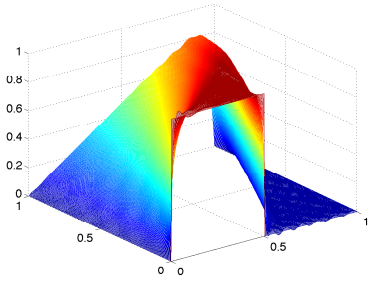
(b) POD (FEM) $\ell = 5$



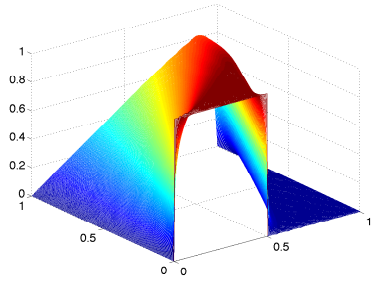
(c) POD (FEM) $\ell = 8$



(d) POD (FEM) $\ell = 11$

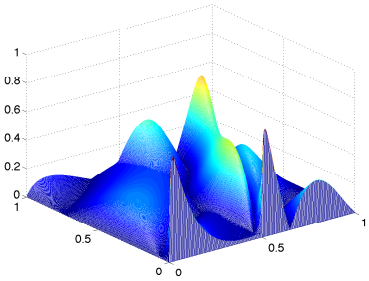


(e) POD (FEM) $\ell = 14$

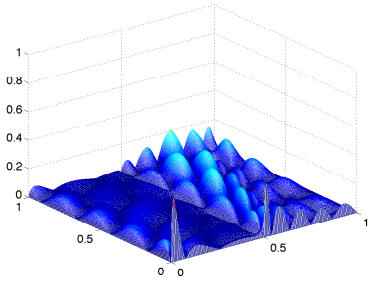


(f) POD (FEM) $\ell = 17$

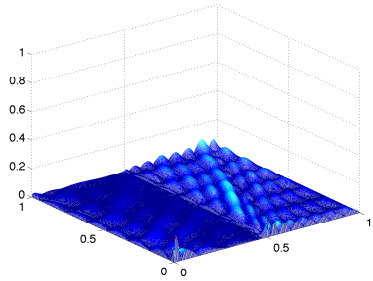
Figure 3.14: Example 2: POD (FEM) approximation



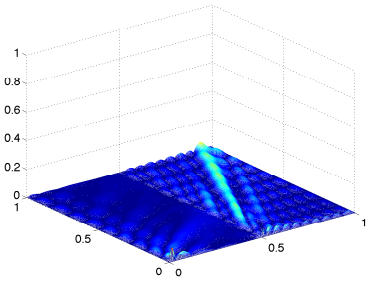
(a) Absolute error $\ell = 2$



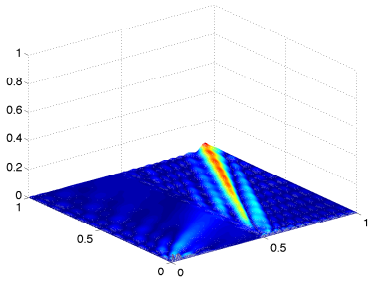
(b) Absolute error $\ell = 5$



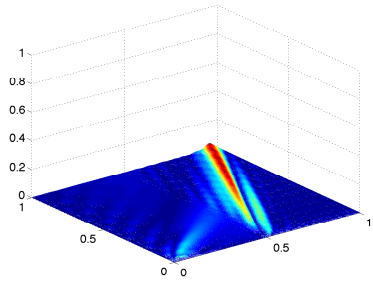
(c) Absolute error $\ell = 8$



(d) Absolute error $\ell = 11$



(e) Absolute error $\ell = 14$



(f) Absolute error $\ell = 17$

Figure 3.15: Example 2: Absolute error of the POD (FEM) approximation

		POD (FD)		POD (FEM)		GPOD	
		$[0, 0.5]$	$[0.5, 1]$	$[0, 0.5]$	$[0.5, 1]$	$[0, 0.5]$	$[0.5, 1]$
Euler	$l = 2$	0.006004	0.017863	0.005839	0.017518	0.006042	0.017701
	$l = 4$	0.000874	0.004523	0.000885	0.004384	0.000891	0.004423
	$l = 6$	0.000193	0.001004	0.000205	0.001025	0.000203	0.000964
	$l = 8$	0.000041	0.000239	0.000047	0.000258	0.000044	0.000218
	$l = 10$	0.000015	0.000109	0.000018	0.000139	0.000016	0.000099
	$l = 12$	0.000009	0.000072	0.000011	0.000107	0.000009	0.000067
	$l = 14$	0.000006	0.000061	0.000008	0.000098	0.000007	0.000059

Table 3.4: L^2 error for the intervals $[0, 0.5]$ and $[0.5, 1]$

3.6.3 DEIM

The error for a different number of DEIM interpolation points is plotted against the number of POD modes in Figure 3.17. The data is generated with the ODE45 solver with the same setting as in Subsection 3.6.2. The error for the POD-DEIM approximation is dominated by the number of POD modes. Increasing only the number of DEIM interpolation points does not give an advantage, since the error obtained from POD dominates the DEIM error. On the other side, increasing the number of POD modes by a fixed interpolation points yield the same result. The error is dominated by the error obtained from DEIM. Hence, the number of POD modes and the number of interpolation points have to be increased simultaneously. According to the conducted experiment, the number of DEIM interpolation points has to exceed the number of POD modes. The plots in Figure 3.18 illustrate this behaviour for a growing number of interpolation points with fixed number of POD modes.

Figure 3.19 illustrates the distribution of the DEIM interpolation points. The x axis denotes the number of the spatial discretization point. Most points are located in the right half side of the plot, which corresponds to the interval $[0.5, 1]$. Recall that this part of the POD basis behaves like a Fourier basis and the resulting error in this interval exceeds the error in the left half interval. To this end, interpolation at this points is important in order to get good approximations.

Finally, Table 3.6 and Table 3.7 present the time measurements and corresponding speedups for the POD-DEIM approximation. The results in Table 3.6 are based on the semi-implict Euler, while the times in Table 3.7 result from integrating the ROM with the ODE45 solver.

For a fast overview of the results, the reader is referred to Figure 3.20 and Figure 3.21. Firstly, the GPOD-DEIM approximation is computationally faster (significant) than the POD-DEIM (FD) approach. The POD-DEIM (FEM) approach is not considered for its more complex nonlinear term, which results in larger times anyway. The speedup compared to the full system almost only depends on the employed number of POD modes. Some inconsistant time measurements appear for small number of interpolation points k. This might be based on internal MATLAB® routines. Again, the semi-implicit Euler is faster than the ODE45 solver, but as mentioned in Subsection 3.6.2 the tradeoff is the error. Overall, the

		POD (FD)		POD (FEM)		GPOD	
		time	speedup	time	speedup	time	speedup
semi-implicit Euler	$l = 2$	0.012311	14.287963	0.021321	12.873413	0.019471	9.077683
	$l = 4$	0.015984	11.005148	0.021136	12.985777	0.020922	8.448093
	$l = 6$	0.017359	10.133174	0.020446	13.423963	0.024438	7.232713
	$l = 8$	0.016549	10.629162	0.022337	12.287865	0.022319	7.919455
	$l = 10$	0.018585	9.465206	0.023466	11.696194	0.026638	6.635498
	$l = 12$	0.019941	8.821521	0.026411	10.392394	0.028515	6.198584
	$l = 14$	0.022998	7.648877	0.030781	8.916902	0.031454	5.619505
	$l = 16$	0.025400	6.925479	0.028175	9.741583	0.031710	5.574050
	$l = 18$	0.025805	6.816786	0.030747	8.926783	0.032743	5.398265
	$l = 20$	0.026211	6.711112	0.033125	8.285835	0.033155	5.331150
ode45	$l = 2$	0.021620	4.948369	0.020937	19.840360	0.020408	18.844952
	$l = 4$	0.017979	5.950375	0.018492	22.463793	0.018183	21.150576
	$l = 6$	0.023267	4.598194	0.024717	16.805816	0.024956	15.410272
	$l = 8$	0.032106	3.332194	0.033751	12.307500	0.034312	11.208177
	$l = 10$	0.040747	2.625587	0.043235	9.607765	0.044938	8.558002
	$l = 12$	0.059069	1.811178	0.059706	6.957266	0.061543	6.248909
	$l = 14$	0.068438	1.563239	0.102411	4.056155	0.107873	3.565118
	$l = 16$	0.076723	1.394433	0.114103	3.640519	0.134179	2.866170
	$l = 18$	0.083805	1.276585	0.148300	2.801032	0.178836	2.150453
	$l = 20$	0.088767	1.205227	0.169924	2.444590	0.197055	1.951638
ode15s	$l = 2$	0.030481	3.397732	0.030787	4.069631	0.032280	3.471235
	$l = 4$	0.038571	2.685085	0.037919	3.304246	0.039236	2.855856
	$l = 6$	0.053312	1.942621	0.054741	2.288847	0.053796	2.082880
	$l = 8$	0.066200	1.564444	0.063210	1.982170	0.065921	1.699787
	$l = 10$	0.071637	1.445702	0.074026	1.692566	0.079528	1.408947
	$l = 12$	0.086686	1.194721	0.090650	1.382168	0.090751	1.234705
	$l = 14$	0.096890	1.068901	0.095142	1.316914	0.107037	1.046844
	$l = 16$	0.108359	0.955759	0.100338	1.248716	0.107829	1.039158
	$l = 18$	0.116098	0.892055	0.108132	1.158712	0.116326	0.963249
	$l = 20$	0.130503	0.793588	0.111180	1.126939	0.118894	0.942445
ode23s	$l = 2$	0.024137	71.938113	0.067653	43.223183	0.025565	100.455663
	$l = 4$	0.059661	29.104325	0.091182	32.069523	0.061551	41.722946
	$l = 6$	0.119903	14.481705	0.181274	16.131284	0.136229	18.851315
	$l = 8$	0.199338	8.710806	0.284695	10.271239	0.228452	11.241320
	$l = 10$	0.288987	6.008562	0.361961	8.078710	0.360191	7.129841
	$l = 12$	0.400266	4.338104	0.426690	6.853164	0.485302	5.291758
	$l = 14$	0.522898	3.320719	0.535825	5.457332	0.657992	3.902940
	$l = 16$	0.637269	2.724749	0.618266	4.729639	0.743012	3.456343
	$l = 18$	0.800272	2.169758	0.710818	4.113816	1.065124	2.411083
	$l = 20$	1.187404	1.462347	0.776977	3.763527	1.188571	2.160664

Table 3.5: Example 2: Computational time and speedup for different numbers ℓ of POD basis functions

		POD-DEIM (FD)		**GPOD-DEIM**	
		time	speedup	time	speedup
$\ell = 5$	$k = 1$	0.010047	14.752813	0.006789	20.358894
	$k = 5$	0.008183	18.114148	0.006973	19.821035
	$k = 9$	0.008538	17.360824	0.006469	21.365987
	$k = 13$	0.010413	14.234842	0.006433	21.484866
	$k = 17$	0.007754	19.115819	0.006495	21.279699
	$k = 21$	0.007897	18.771121	0.006553	21.090859
	$k = 25$	0.007893	18.778726	0.006387	21.638544
$\ell = 9$	$k = 1$	0.009767	15.176595	0.007616	18.147896
	$k = 5$	0.008316	17.823519	0.007103	19.456934
	$k = 9$	0.009014	16.445022	0.007159	19.305285
	$k = 13$	0.009908	14.960170	0.007171	19.274159
	$k = 17$	0.009051	16.377479	0.007162	19.297363
	$k = 21$	0.008721	16.996563	0.007272	19.005710
	$k = 25$	0.010844	13.668555	0.007184	19.237705
$\ell = 13$	$k = 1$	0.012713	11.659370	0.009798	14.105944
	$k = 5$	0.010136	14.624506	0.009482	14.575498
	$k = 9$	0.010492	14.128307	0.009380	14.734177
	$k = 13$	0.010879	13.625258	0.009375	14.742880
	$k = 17$	0.012247	12.103240	0.009423	14.667098
	$k = 21$	0.011208	13.225267	0.009368	14.753038
	$k = 25$	0.011183	13.255249	0.009537	14.491698
$\ell = 17$	$k = 1$	0.013536	10.950653	0.010666	12.957629
	$k = 5$	0.012242	12.107814	0.010510	13.149754
	$k = 9$	0.011876	12.480931	0.010426	13.256158
	$k = 13$	0.013868	10.688376	0.010651	12.976310
	$k = 17$	0.012873	11.514566	0.010882	12.700651
	$k = 21$	0.012225	12.125062	0.010744	12.863660
	$k = 25$	0.011795	12.566579	0.011254	12.280470
$\ell = 21$	$k = 1$	0.015691	9.446842	0.012303	11.233538
	$k = 5$	0.015737	9.418805	0.012385	11.159145
	$k = 9$	0.013917	10.650713	0.012406	11.140068
	$k = 13$	0.013429	11.038246	0.012401	11.144872
	$k = 17$	0.015218	9.740200	0.012601	10.967586
	$k = 21$	0.015676	9.455969	0.012429	11.119536
	$k = 25$	0.015039	9.856145	0.012382	11.161778

Table 3.6: Example 2: Computational time and speedup for different numbers of POD modes and DEIM interpolation points. Solver: semi-implict Euler

		POD-DEIM (FD)		**GPOD-DEIM**	
		time	speedup	time	speedup
$\ell = 5$	$k = 1$	0.031209	3.079049	0.017560	19.654320
	$k = 5$	0.021645	4.439537	0.015595	22.130879
	$k = 9$	0.018478	5.200429	0.012085	28.558219
	$k = 13$	0.016574	5.797631	0.012415	27.799353
	$k = 17$	0.018877	5.090560	0.012285	28.093666
	$k = 21$	0.019548	4.915639	0.012629	27.327296
	$k = 25$	0.019468	4.935957	0.012419	27.790128
$\ell = 9$	$k = 1$	0.031107	3.089055	0.017441	19.787889
	$k = 5$	0.028097	3.420008	0.021305	16.198911
	$k = 9$	0.026220	3.664837	0.021827	15.811960
	$k = 13$	0.025958	3.701887	0.020953	16.470913
	$k = 17$	0.025197	3.813584	0.021051	16.394324
	$k = 21$	0.026593	3.613518	0.021040	16.403026
	$k = 25$	0.025550	3.760952	0.021153	16.315805
$\ell = 13$	$k = 1$	0.063083	1.523262	0.034929	9.880746
	$k = 5$	0.045545	2.109820	0.038989	8.851634
	$k = 9$	0.042027	2.286449	0.045347	7.610655
	$k = 13$	0.038144	2.519213	0.047054	7.334543
	$k = 17$	0.044009	2.183486	0.046102	7.486096
	$k = 21$	0.045691	2.103094	0.044922	7.682613
	$k = 25$	0.047449	2.025164	0.043527	7.928967
$\ell = 17$	$k = 1$	0.053788	1.786491	0.070248	4.912858
	$k = 5$	0.057631	1.667378	0.069742	4.948518
	$k = 9$	0.061551	1.561176	0.071768	4.808819
	$k = 13$	0.055384	1.735023	0.073476	4.697047
	$k = 17$	0.063187	1.520753	0.076267	4.525141
	$k = 21$	0.056502	1.700681	0.073751	4.679511
	$k = 25$	0.054371	1.767350	0.074052	4.660498
$\ell = 21$	$k = 1$	0.062114	1.547027	0.100453	3.435641
	$k = 5$	0.063358	1.516655	0.100524	3.433215
	$k = 9$	0.070539	1.362265	0.099466	3.469721
	$k = 13$	0.064095	1.499232	0.099217	3.478445
	$k = 17$	0.069549	1.381661	0.099227	3.478091
	$k = 21$	0.065841	1.459469	0.100704	3.427092
	$k = 25$	0.062978	1.525808	0.100410	3.437113

Table 3.7: Example 2: Computational time and speedup for different numbers of POD modes and DEIM interpolation points. Solver: ODE45

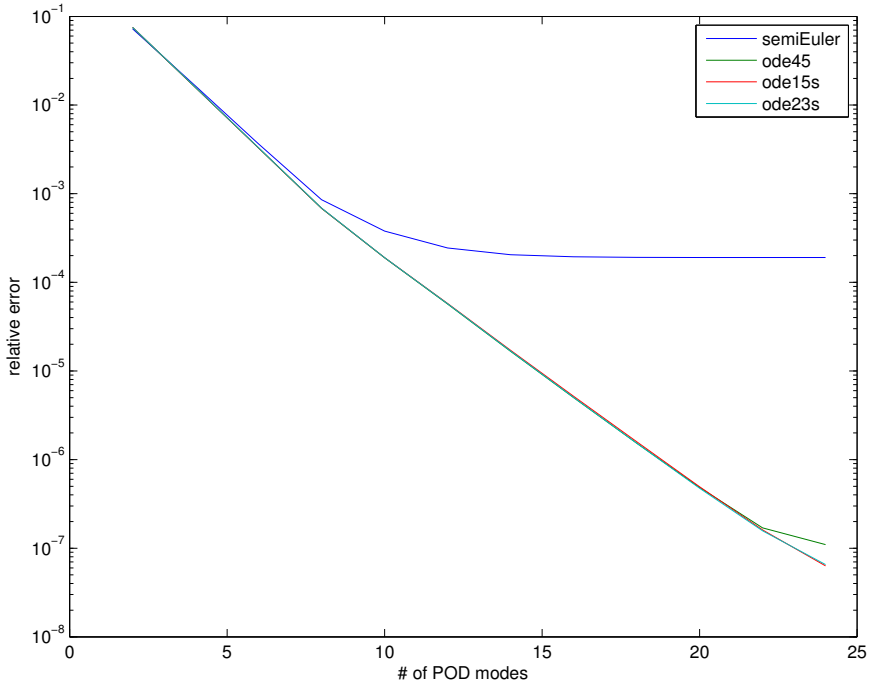


Figure 3.16: Example 2: POD (FD) relative error

GPOD-DEIM approach is more than ten times faster than the full system and still yields a good approximation. The POD-DEIM (FD) approach is only slightly slower and also a reasonable choice for a ROM.

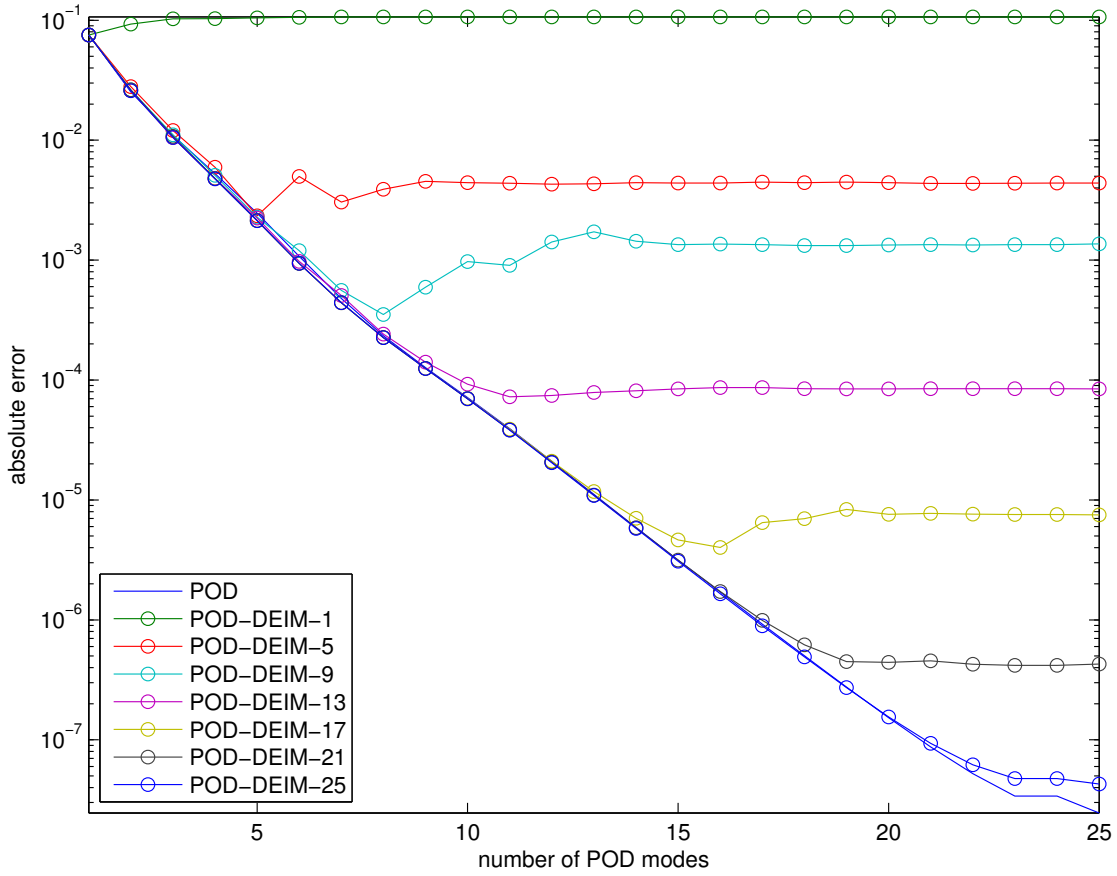
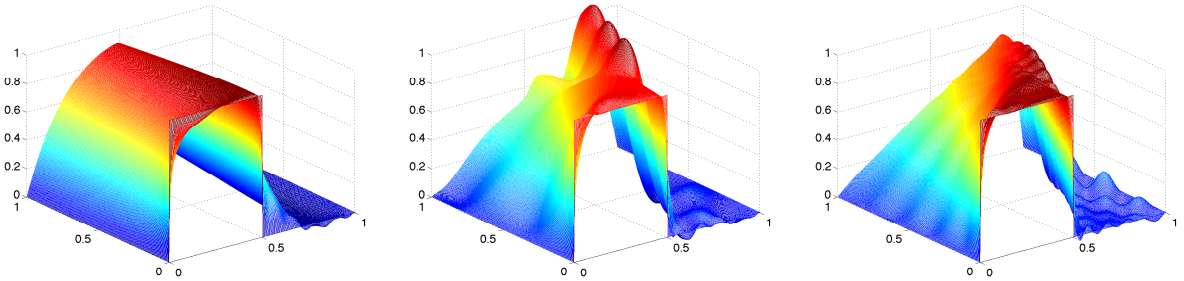
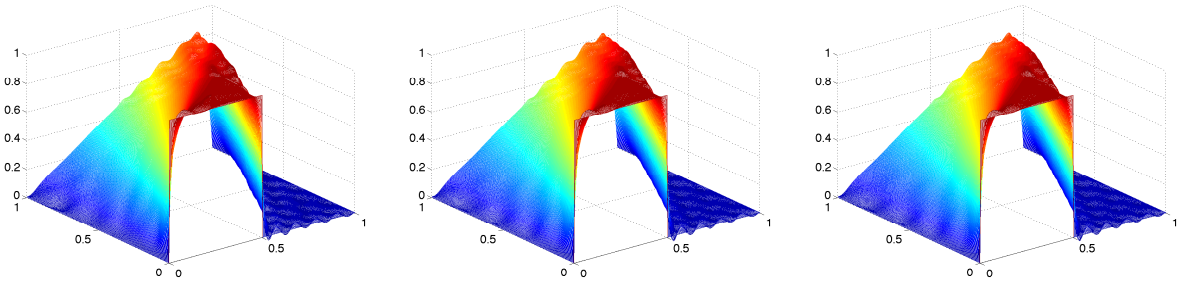


Figure 3.17: Error of the POD-DEIM (FD) approximation



(a) $k = 1$ (b) $k = 5$ (c) $k = 9$



(d) $k = 13$ (e) $k = 17$ (f) $k = 21$

Figure 3.18: The POD-DEIM approximation with $\ell = 11$ POD modes and varying numbers of interpolation points k

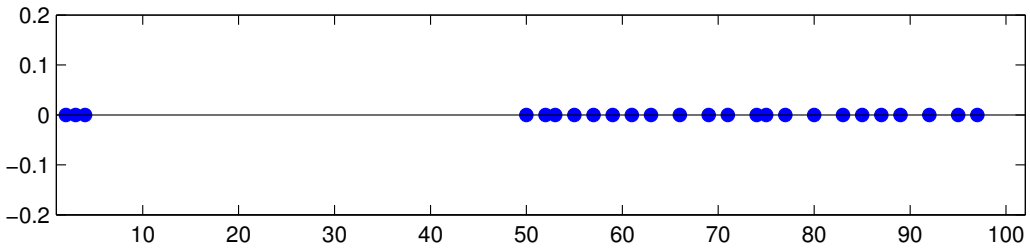
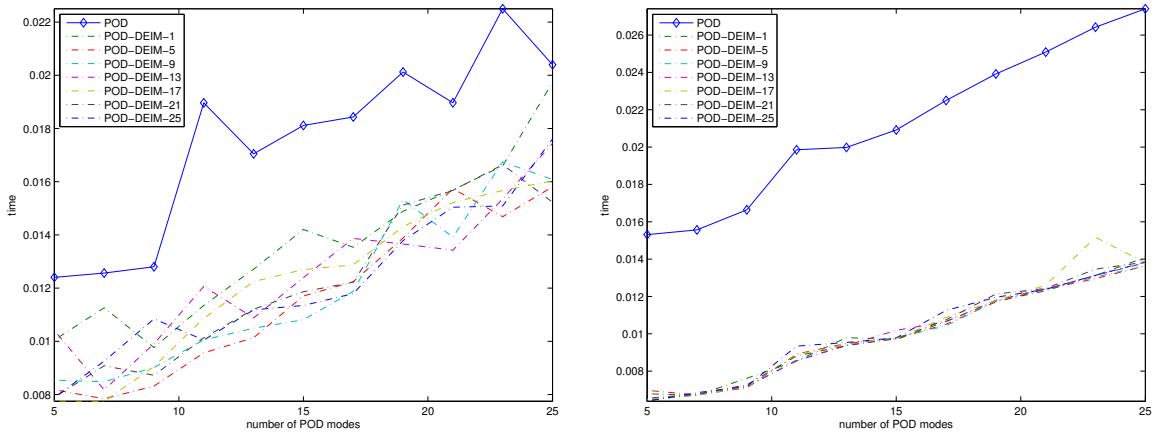
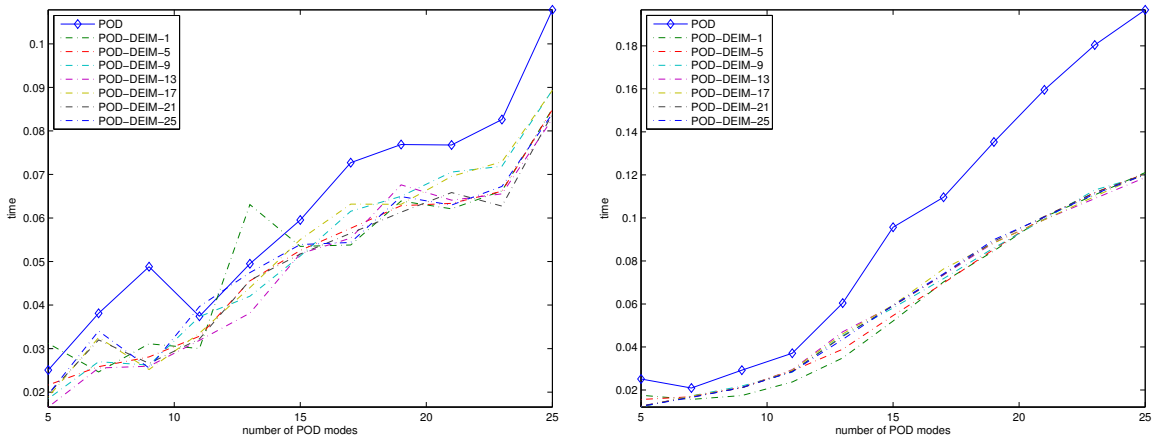


Figure 3.19: DEIM interpolation points



(a) POD-DEIM (FD) (b) GPOD-DEIM

Figure 3.20: Computational time of POD-DEIM (FD) and GPOD-DEIM. Solver: semi-implicit Euler



(a) POD-DEIM (FD) (b) GPOD-DEIM

Figure 3.21: Computational time of POD-DEIM (FD) and GPOD-DEIM. Solver: ODE45

Chapter 4

Application: Optimal Control of Burgers' Equation

In the computational fluid dynamics (CFD) community, control of fluid flows is one of the most challenging areas with a lot of work going on. Various numerical methods for control purposes have been established over the last half century, including steepest descent, conjugate gradient, penalty-methods, primal-dual approaches and higher order approximations. The treatment of equality and inequality constraints has been investigated. The interested reader is referred to the references given at the end of Section 4.2. A broad overview of optimal control topics is given in [28] and the references therein. Numerical treatment is expensive due to the large amount of linear and nonlinear solves required by all such approaches. ROM seems to be a fruitful approach to adapt these approaches with lower computational cost, making control methods numerically feasible. Several researchers have investigated POD in terms of optimal control, for example Afanasiev and Hinze [1], Arian, Fahl and Sachs [5], Atwell and King [7], Kunisch and Volkwein [44] and Agarwal and Biegler [2]. An error estimate for POD reduced models in control is given in [33].

4.1 Problem Formulation

As a test problem for this work we consider the semilinear quadratic regulator (sLQR) problem:

$$\min \mathcal{J}(u) = \frac{1}{2}\int_0^T \left(\int_0^1 |y(t,x) - y^d(t,x)|^2 dx\right) dt + \frac{\beta}{2}\int_0^T \left(\int_0^1 |u(x,t)|^2 dx\right) dt \tag{4.1}$$

subject to

$$y_t + yy_x - \nu y_{xx} = u \qquad \text{on } (0,T)\times[0,1], \tag{4.2a}$$
$$y(t,0) = y(t,1) = 0 \qquad \text{for a.e. } t \in (0,T), \tag{4.2b}$$
$$y(0,x) = y_0(x) \qquad \text{for all } x \in [0,1]. \tag{4.2c}$$

Here, $y^d : [0,T] \times [0,1] \to \mathbb{R}$ is the *desired state* and $\beta > 0$ a constant weighting the cost of a control action. We call $u : (0,T) \times [0,1] \to \mathbb{R}$ the *control* or *input* and y the *state*. Since the control acts as a forcing term over the whole domain $(0,T) \times [0,1]$ this is a distributed control problem. The name sLQR arises, since the quadratic cost functional $\mathcal{J}$ is constrained to the semilinear PDE (4.2). The theory for LQR problems is well funded and has various approaches including the Pontryagin's maximum principle or Dynamic Programming. This leads to several existence and uniqueness properties. However, for a nonlinear side constraint, we still lack these properties in general. Sometimes is is useful to have a further constraint on the control, in the easiest case a box constraint, which might lead to Bang-Bang controls. For this work, we do not put any restrictions on the control but measure its energy in the cost functional.

4.2 Gradient-Based Optimization

In theory, optimal solutions can be obtained via the Lagrange multiplier approach. This approach yields an optimality system that has to be solved. This so called one-shot approach requires the solution of a large (coupled) system. Sometimes it might be possible to decouple the system, however for nonlinear side constraints this is not possible in general and therefore not desirable. For more details on the one-shot approach the reader is referred to [28] and the references within. Many modern optimization algorithms are gradient based methods. One such algorithm to find a (local) minimum is the method of steepest descent. Starting with some initial data (starting point) we compute the direction of the steepest descent. The steepest descent is given by the negative gradient. To see this, consider a smooth function $g : \mathbb{R}^N \to \mathbb{R}$. From the Taylor expansion we know that

$$g(x_0 + \Delta x) \approx g(x_0) + \nabla g(x_0)^T \Delta x.$$

In order to find the (locally) steepest descent direction Δx we have to solve the optimization problem

$$\Delta x = \arg \min_{\|\Delta x\|=1} g(x_0) + \nabla g(x_0)^T \Delta x = \arg \min_{\|\Delta x\|=1} \nabla g(x_0)^T \Delta x.$$

The minimum is achieved by the negative gradient, that is

$$\Delta x^* = -\nabla g(x_0).$$

The next question is how far to go in the descent direction, or in other words find an optimal step size. This is necessary since the gradient is only a local information and henceforth we might have to update the descent direction. The optimal step size is given by the solution of the optimization problem

$$s = \arg \min_{s \in (0,1)} g(x_0 + s(\Delta x^* - x_0)). \tag{4.3}$$

Solving (4.3) might be quite expensive. A common workaround is an approximation using a line-search algorithm. A discussion on such methods is given in Section 4.5.

For our discussion, we omit the analysis and start directly with the discretized version of the optimal control problem. Note that we can use the matrices $\mathbf{T}_s$ and $\mathbf{T}_t$ developed in Section 2.6 to approximate the integrals in the cost functional $\mathcal{J}$. The discretization of the side constraint in form of Burgers' equation was discussed in detail in Chapter 2. We refer to the discretized problem as $\mathcal{J}_D$ to make notation more obvious. Summarizing the previous discussion leads to Algorithm 5, which is known as the method of *steepest descent*.

Algorithm 5 Steepest Descent

Input: initial control u_0
Output: optimal (locally) control u
Initialize $n = 0$

1. Find descent direction Δu_n by computing the gradient

$$\Delta u_n = \nabla \mathcal{J}_D(u_n). \tag{4.4}$$

2. Optimize along Δu_n, that is find step size s_n by solving the minimization problem

$$s_n = \arg \min_{s \in (0,1]} \mathcal{J}_D(u_n - s(\Delta u_n - u_n)) \tag{4.5}$$

3. Update $u_{n+1} \leftarrow u_n + s_n(v_n - u_n)$,
   $n \leftarrow n + 1$

To analyze the cost of Algorithm 5 we first mention that a simple evaluation of the cost functional $\mathcal{J}$ involves the full computation of the side constraint, that is a full solution of a nonlinear PDE, namely Burgers' equation. Henceforth, every cost function evaluation is expensive. In particular, solving the optimization problem (4.5) is costly and furthermore, computing the gradient (4.4) is quite delicate. More details on the gradient computation is given in Section 4.4.

It shall not be concealed that the steepest descent algorithm is not state of the art. More sophisticated algorithms are for example the conjugate gradient method [45], [54], Newton's method, for example [42] or primal-dual algorithms, for example [9], [10] and [32] or further penalty methods [46]. A further approximation scheme is based on the work of Burns and Cliff [15].

4.3 Trust-Region POD

This sections outlines the Trust-Region Proper Orthogonal Decomposition algorithm proposed 2000 by Arian, Fahl and Sachs [5]. In the previous section we already investigated the cost of the steepest descent method. Hence it is fairly reasonable to do the computations for (4.4) and (4.5) with a reduced model. In their paper they use POD to derive the reduced

model and in order to guarantee convergence tne authors embed the method of steepest descent in a trust-region framework. Let $m_k = \mathcal{J}_D^{\text{POD}}$ denote the cost function for the reduced model based on the snapshots corresponding to the control u_k. Note, that the computation is still the same, the only difference is that we replace y by the low-rank approximation y^{POD}. The procedure is outlined in Algorithm 6.

Algorithm 6 Trust-Region Proper Orthogonal Decomposition

Input: initial control u_0, initial trust-region radius δ_0 and trust-region parameters $0 < \eta_1 < \eta_2 < 1$ and $0 < \gamma_1 \leq \gamma_2 < 1 \leq \gamma_3,\ 0 < \gamma_4 < 1$.
Output: optimal (locally) control u
Initialize: $k = 0$.

1. Compute the solution of the side constraint (full model) with the control u_k.
2. Compute the POD basis and build the corresponding reduced model m_k.
3. Compute the gradient $g_k := \nabla m_k(u_k)$ of the reduced model.
4. Minimize the reduced cost function along g_k within the trust-region radius, i.e. find the stepsize s_k by solving

$$s_k = \arg\min_{s \in \mathbb{R}} m_k(u_k - s g_k) \qquad \text{subject to} \qquad |s| \|g_k\| \leq \delta_k \tag{4.6}$$

5. Compute $\mathcal{J}_D(u_k - s_k g_k)$ and set

$$\rho_k = \frac{\mathcal{J}_D(u_k) - \mathcal{J}_D(u_k - s_k g_k)}{m_k(u_k) - m_k(u_k - s_k g_k)}.$$

6. Update the trust-region radius:
   - If $\rho_k \geq \eta_2$: Set $u_{k+1} = u_k - s_k g_k$ and increase the trust-region radius $\delta_{k+1} = \gamma_3 \delta_k$, set $k = k + 1$ and go back to 1.
   - If $\eta_1 < \rho_k < \eta_2$: Set $u_{k+1} = u_k - s_k g_k$ and decrease the trust-region radius $\delta_{k+1} = \gamma_2 \delta_k$, set $k = k + 1$ and go back to 1.
   - If $\rho_k \leq \eta_1$: Decrease the trust-region radius $\delta_{k+1} = \min\{\gamma_4 |s_k|, \gamma_1 \delta_k\}$, set $k = k+1$ and go to step 3.

Based on [67], we give the following definition. An iteration is called *successful* if $\rho_k > \eta_1$, i.e. if the ratio of the actual and the predicted reduction is large enough. Accordingly, the iteration is *unsuccessful* if $\rho_k \leq \eta_1$. Using some standard assumptions on the cost functional $\mathcal{J}$ we can guarantee convergence with the following theorem. For a proof, see [5].

Theorem 4.1 (Convergence of TRPOD)**.** *Suppose $\mathcal{J}_D$ is regular enough and $g_k := \nabla m_k$ satisfies*

$$\frac{\|g_k - \nabla \mathcal{J}_D(u_k)\|}{\|g_k\|} \leq \xi \qquad \textit{for all } k \geq N$$

for some $0 < \xi < 1 - \eta_2$. *Then*

$$\liminf_{k\to\infty} \|g_k\| = 0.$$

In the discussion of the ROM we have already outlined that the nonlinearity is still expensive in the POD-based reduced model. Therefore it is a natural idea to expand the TRPOD in terms of DEIM. That means that the reduced model $m_k = \mathcal{J}_D^{\text{POD}}$ is replaced by

$$m_k^{\text{DEIM}} = \mathcal{J}_D^{\text{POD-DEIM}}.$$

Analysis of the convergence proof of the TRPOD does not encounter special POD properties. Furthermore, the proof is only based on inexact gradient information and a non-quadratic cost function. The POD-DEIM procedure enforces just this setting so we can expect a natural extension of the convergence proof.

Implementation strategies The convergence result (4.1) yields the following strategy. In order to converge, the gradient of the reduced model should almost match the one of the full model for all $k \geq N$. However, in the beginning, inexact gradient information are valid. The strategy is to start with very few POD modes and interpolation points and refine the ROM during the algorithm to guarantee convergence.

4.4 Computation of the Gradient

Computing a gradient is a crucial point to various optimization algorithms including the earlier proposed steepest descent method. In terms of the optimal control problem (4.8) this is quite tedious. We present two different approaches for gradient computations. One is based on a finite difference approximation, the other takes advantages from tools of sensitivity analysis, namely the adjoint method.

4.4.1 Finite Differences

A simple and straight forward approach is to employ a finite differences scheme. As we have seen in Section 2.1, the derivative of a smooth function $g : \mathbb{R} \to \mathbb{R}$ can be approximated by using centered finite differences in the form

$$g'(x) \approx \frac{g(x+h) - g(x-h)}{2h} \qquad \text{for } h \ll 1. \tag{4.7}$$

However, in case of our optimal control problem, the cost functional $\mathcal{J}$ depends on the control u, which is in the discretized case an $(N \times M)$ matrix denoted with $\mathbf{U}$. Henceforth we have $\mathcal{J}_D : \mathbb{R}^{N\times M} \to \mathbb{R}$ and the gradient is given by

$$\nabla \mathcal{J}_D(\mathbf{U}) = \begin{pmatrix} \frac{\partial}{\partial u_{11}}\mathcal{J}_D & \frac{\partial}{\partial u_{12}}\mathcal{J}_D & \cdots & \frac{\partial}{\partial u_{1M}}\mathcal{J}_D \\ \frac{\partial}{\partial u_{21}}\mathcal{J}_D & \frac{\partial}{\partial u_{22}}\mathcal{J}_D & \cdots & \frac{\partial}{\partial u_{2M}}\mathcal{J}_D \\ \vdots & \vdots & & \vdots \\ \frac{\partial}{\partial u_{N1}}\mathcal{J}_D & \frac{\partial}{\partial u_{N2}}\mathcal{J}_D & \cdots & \frac{\partial}{\partial u_{NM}}\mathcal{J}_D \end{pmatrix} (\mathbf{U}) \in \mathbb{R}^{N\times M}.$$

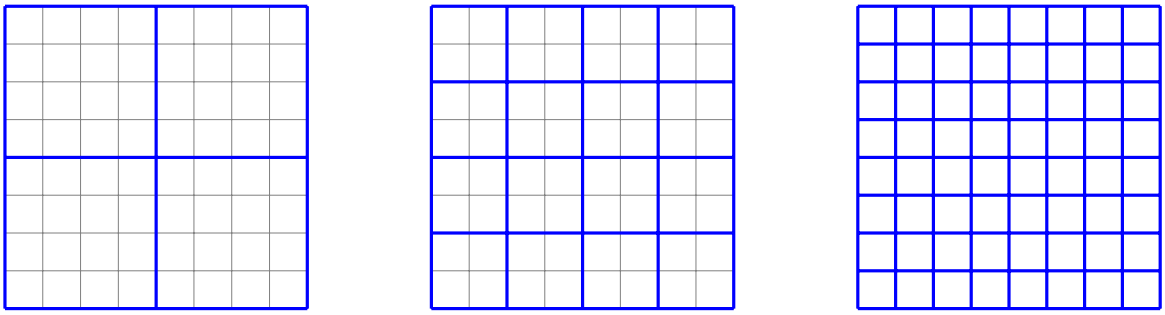


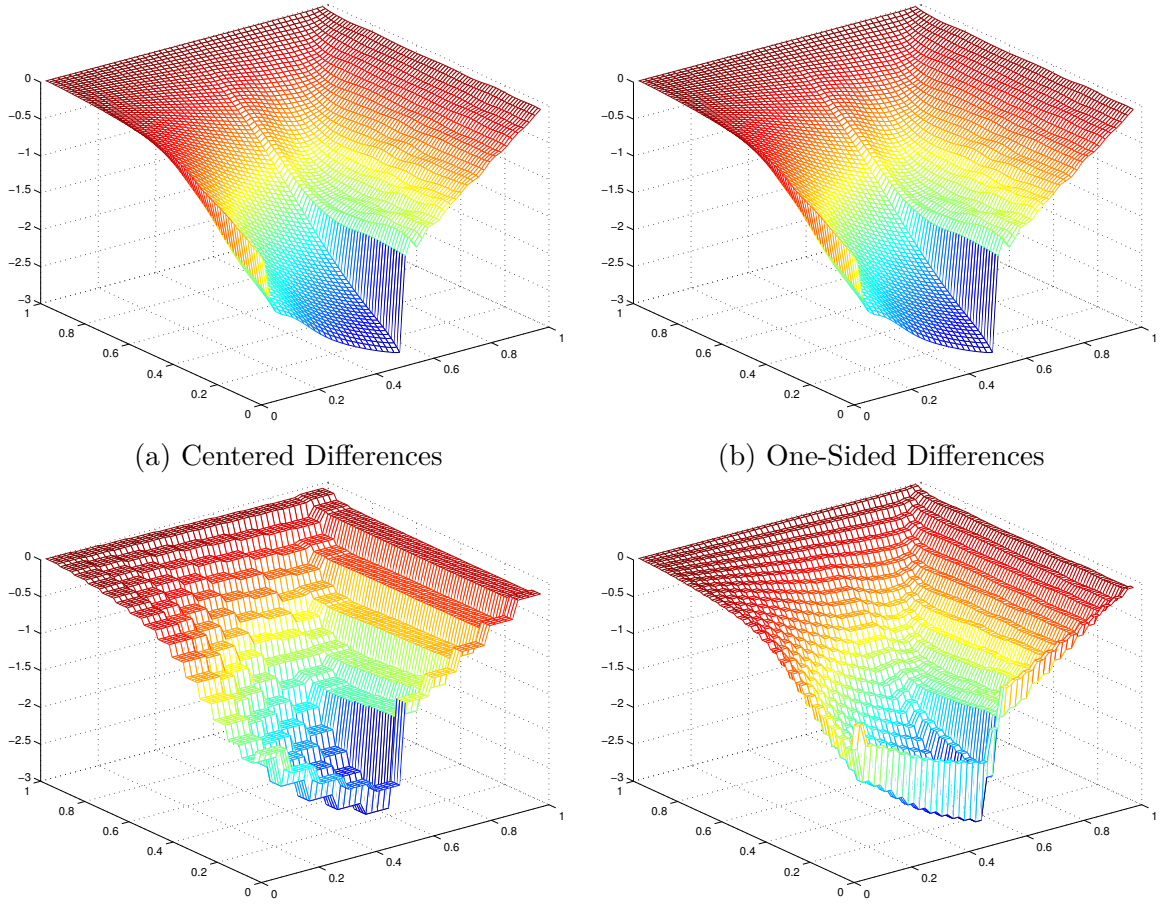
Figure 4.1: Multilevel clustering approach for the gradient

Applying the finite difference approximation here, means that we have $2NM$ evaluations of $\mathcal{J}_D$ that involve the integration of the PDE. Even for a coarse discretization, this is very expensive. For example, if $N = 30$ and $M = 50$ we have a total of 3000 cost function evaluations only to compute the gradient. Even with a reduced model this is expensive. The first strategy to avoid evaluations is to use one-sided differences instead of centered differences. A one-sided approximation of the derivative is given by

$$g'(x) \approx \frac{g(x+h) - g(x)}{h} \qquad \text{for } h \ll 1.$$

This approximation is in the class $\mathcal{O}(h)$ while (4.7) lies in $\mathcal{O}(h^2)$. However, we are using inexact gradient information anyways, so there is no reason for the "better" approximation (4.7) as long as we can guarantee a sufficiently good approximation close to the optimum. This leads directly to a multilevel approach. Instead of computing the derivate for each component of the gradient (in Figure 4.1 denoted by a square), we employ a clustering strategy. This clustered information can be seen as an average gradient over this cluster. Even for small clusters, this saves many cost function evaluations. The multilevel strategy is to start with a coarse cluster and refine it during the algorithm.

The different approaches are presented in Figure 4.2. For $N = 50$ and $M = 80$ discretization points in space and time, respectively, centered differences, one-sided differences and two cluster strategies are employed. All computations are based on the ROM (POD modes:$\ell = 15$, DEIM interpolation points: $k = 23$) for Example 2. The reader shoul observe that the centered differences approach yields almost identical results compared to the one-sided differences approach. From theory, the centered differences approach is superior, since the error is in $\mathcal{O}(h^2)$ compared to $\mathcal{O}(h)$ for the one-sided differences. However, since the cost functional is almost quadratic, the second derivative almost vanishes and henceforth the error constant is negligible. To evaluate the cluster strategy, we use the following notation. A $a \times b$ cluster, means a cells in spatial direction and b cells in temporal direction. Both employed clusters (5×5 and 2×3) yield gradients of the same shape as the classical finite differences approach.



(a) Centered Differences

(b) One-Sided Differences

(c) Cluster Strategy: 5×5 cluster

(d) Cluster Strategy: 2×3 cluster

Figure 4.2: Negative normalized gradient based on the ROM for Example 2

4.4.2 Adjoint Method

A more sophisticated method to compute the gradient is based on the methods of sensitivity analysis - the so called *adjoint method*. We give a short introduction fitted to our problem setting which is based on [13]. For more information on the adjoint method see for example [24]. Consider the general problem

$$\min_u \mathcal{G}(y,u) = \int_0^T g(t,y,u)dt \qquad \text{subject to} \qquad \begin{cases} e_1(t,y,\dot{y},u) = 0, \\ e_2(y(0),u) = 0. \end{cases} \tag{4.8}$$

Here, $e_1(t,y,\dot{y},u) = 0$ is an ODE in implicit form with initial condition given by e_2. We assume, that the functions g, e_1 and e_2 are at least continuously differentiable. We first note, that every explicit ODE of the form

$$\dot{x} = h(t,y,u)$$

can be written in implicit form as $e_1(t,y,\dot{y},u) = \dot{y} - h(t,y,u) = 0$. Moreover, this relationship reveals the u dependence of the state variable y. Hence, a more precise notation would be

$\mathcal{G}(y(u),u)$ but for convenience, we suppress the argument. It is easy to see that the discretized $\mathcal{J}_D$ is a special case of (4.8) that also satisfies the smoothness assumptions and hence this approach is suitable for our problem. The gradient is given by the total derivative

$$d_u\mathcal{G}(y,u) = \int_0^T (\partial_y g \; d_u y + \partial_u g)\, dt. \tag{4.9}$$

The challenging component here is the derivative of the state variable $d_u y$. The adjoint method avoids its computation by a clever choice of Lagrangian multipliers. Similar to the derivation of the POD basis, the first step is the introduction of the Lagrange function $\mathcal{L} : [0,T] \times L^2([0,T]\times[0,1]) \times L^2([0,T]\times[0,1]) \times L^2([0,T]) \times \mathbb{R} \to \mathbb{R}$

$$\mathcal{L}(t,y,u,\lambda,\eta) = \mathcal{G}(y,u) + \int_0^T \lambda^\top e_1(t,y,\dot{y},u)dt + \eta^\top e_2(y(0),u),$$

with time-dependent vector of Langrange multiplier λ and additional vector of Lagrange multipliers η. For the derivation we assume, that the side constraints e_1 and e_2 are always satisfied. This implies $\mathcal{G}(y,u) \equiv \mathcal{L}(t,y,u,\lambda,\eta)$. In particular, we have

$$d_u\mathcal{G}(y,u) \equiv d_u\mathcal{L}(t,y,u,\lambda,\eta).$$

Taking the total derivative of the Lagrangian yields

$$\begin{aligned} d_u\mathcal{L} = &\int_0^T \left(\partial_y g \; d_u y + \partial_u g + \lambda^\top \left(\partial_y e_1 \; d_u y + \partial_{\dot{y}} e_1 \; d_u \dot{y} + \partial_u e_1\right)\right) dt \\ &+ \eta^\top \left(\partial_{y(0)} e_2 \; d_u y(0) + \partial_u e_2\right). \end{aligned} \tag{4.10}$$

The aim now is to eliminate the derivatives related to the state variable. First, we integrate by parts to obtain

$$\int_0^T \lambda^\top \partial_{\dot{y}} e_1 \; d_u \dot{y} dt = \lambda^\top \partial_{\dot{y}} e_1 \; d_u y\Big|_0^T - \int_0^T \left(\dot{\lambda}^\top \partial_{\dot{y}} e_1 + \lambda^\top d_t \partial_{\dot{y}} e_1\right) d_u y dt. \tag{4.11}$$

Substitution of (4.11) into (4.10) yields

$$\begin{aligned} d_u\mathcal{L} = &\int_0^T \left(\left[\partial_y g + \lambda^\top \left(\partial_y e_1 - d_t \partial_{\dot{y}} e_1\right) - \dot{\lambda}^\top \partial_{\dot{y}} e_1\right] d_u y + \partial_u g + \lambda^\top \partial_u e_1\right) dt \\ &+ \lambda^\top \partial_{\dot{y}} e_1 d_u y\Big|_0^T + \eta^\top \left(\partial_{y(0)} e_2 \; d_u y(0) + \partial_u e_2\right). \end{aligned}$$

The derivatives of the state variable y can be eliminated with the following setting:

1. Set $\lambda(T) = 0$. Then $\lambda^\top \partial_{\dot{y}} e_1 \; d_u y\Big|^T = 0$.
2. Let $\eta^\top = \lambda^\top \partial_{\dot{y}} e_1\Big|^0 \left(\partial_{y(0)} e_2\right)^{-1}$. This implies

$$-\lambda^\top \partial_{\dot{y}} e_1 \; d_u y\Big|^0 + \eta^\top \partial_{y(0)} e_2 \; d_u y(0) = 0.$$

3. For $t > 0$ let λ satisfy
$$\dot{\lambda}^\top = \lambda^\top \left(d_t \partial_{\dot{y}} e_1 - \partial_y e_1\right) - \partial_y g. \tag{4.12}$$

At this point we give the following definition:

Definition 4.2 (Adjoint Equation)**.** *Equation* (4.12) *is called the* adjoint equation.

For this special choice of λ the total derivative is given by

$$d_u \mathcal{G} = d_u \mathcal{L} = \int_0^T \left(\partial_u g + \lambda^\top \partial_u e_1\right) dt + \lambda^\top \partial_{\dot{y}} e_1 \Big|^0 \left(\partial_{y(0)} e_2\right)^{-1} \partial_u e_2 \tag{4.13}$$

and does no longer depend on derivatives of the state variable. We summarize the results in Algorithm 7.

Algorithm 7 Adjoint Method

Input: control u.
Output: derivate $d_u \mathcal{G}$.

1. Solve $e_1(t, y, \dot{y}, u) = 0$ for y from $t = 0$ to T with initial condition $e_2(y(0), u) = 0$.
2. Solve (4.12) for λ from $t = T$ to 0 with initial condition $\lambda(T) = 0$.
3. Compute (4.13) to obtain the total derivative $d_u \mathcal{G}$.

Before employing the adjoint method on Burgers' equation we give a short glance on the topic differentiate-then-discretize vs. discretize-then-differentiate. The outlook is based on [28]. The above derivation of the adjoint method is based on the implicit ODE as side constraint. However, it can be derived from a general PDE as well. The natural question to ask is weather one should use the discretized ODE to achieve the adjoint equation or find a continuous adjoint equation based on the PDE constraint and the discretize it. This question is known as *differentiate-then discretize* vs. *discretize-then-differentiate*.

Advantages of the discretize-then-differentiate approach: The first point here to mention is the consistency of the gradient. In the differentiate-then-discretize case, the obtained gradient is neither the exact gradient of the continuous cost functional nor of the discretized cost function. As shown in [28, Sec. 4.1.2] this might lead to a negative gradient pointing uphill, which certainly is not a descent direction. On the other side, the gradient obtained by the adjoint of the discretized system yields an exact gradient of the discretized cost functional. Moreover, the adjoint variables can be computed employing automatic differentiation software [36], [51]. Finally, we can also use the ROM system and derive a reduced-order adjoint system, which is fast to solve.

Advantages of the differentiate-then-discretize approach: In the discretize-then-differentiate approach a fixed spatial grid is employed, which is based on the properties of the forward problem. The differentiate-then-discretize approach yields more flexibility and offers adaptive grids in both, the state and the adjoint system. Again, some examples are given in [28]. However, in this thesis we only consider a uniform spatial grid. A second point to mention is the independence of numerical issues within the discretization process such as oscillations or shocks. The linear adjoint equation is usually easier to solve then the nonlinear state equation. It should be noted that one can overcome the disadvantages of the differentiate-then-discretize approach, most of them via regularization techniques and hybrid approaches.

For this work, we consider the first approach. The adjoint system for Burgers' equation of the latter one is for example given in [44]. Henceforth, we conclude this section with the specific functions for the discretized Burgers' equation and the derivation of the corresponding adjoint systems. The functions e_1 and e_2 are given by

$$\begin{aligned} e_1\left(t, \alpha, \dot{\alpha}, u\right) &= \mathbf{M}\dot{\alpha}(t) - \mathbf{A}\alpha(t) - \mathcal{N}\left(\alpha(t)\right) - \mathbf{B}\overline{u}(t), \\ e_2\left(\alpha(0), u\right) &= \alpha(0) - \overline{\alpha} \end{aligned}$$

with partial derivatives

$$\begin{aligned} \partial_y e_1 &= -\mathbf{A} - \nabla\mathcal{N}\left(\alpha(t)\right), & \partial_{y(0)} e_2 &= \mathbf{I}_N, \\ \partial_{\dot{y}} e_1 &= \mathbf{I}_N, & \partial_u e_2 &= \mathbf{0}_N. \\ \partial_u e_1 &= -\mathbf{B}. \end{aligned}$$

The cost function g is given by $g(t, y, u) = \mathcal{J}(y, u)$ and the partial derivatives are given by

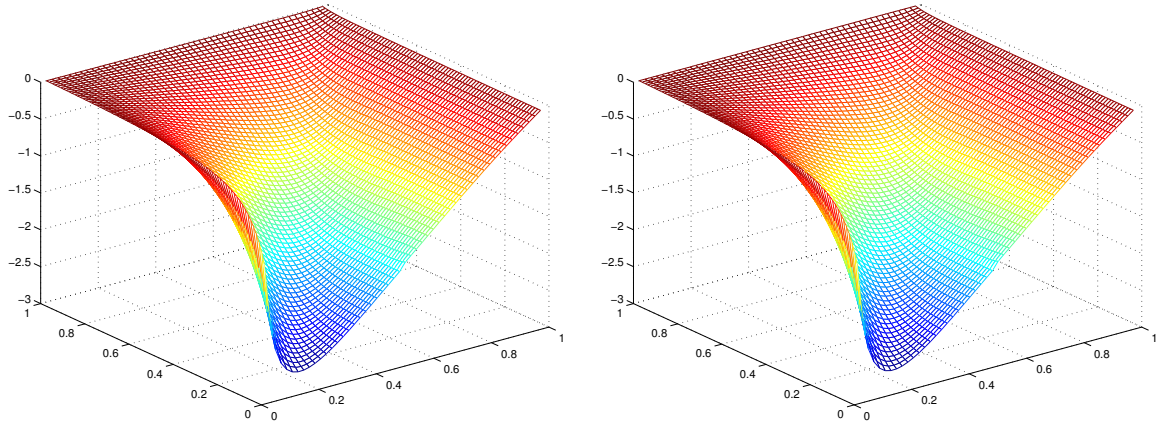
$$\frac{\partial}{\partial u}\mathcal{J} = \beta u^\top \qquad \text{and} \qquad \frac{\partial}{\partial y}\mathcal{J} = \left(y - y^d\right)^\top.$$

Henceforth, the adjoint equation (4.12) reads as

$$\begin{cases} \dot{\lambda} = \mathbf{A}^\top\lambda + \left(\nabla\mathcal{N}\left(\alpha(t)\right)\right)^\top\lambda - \left(y - y^d\right) \\ \lambda(T) = 0 \end{cases} \tag{4.14}$$

and has to be solved backwards in time. This is the reason that the adjoint problem is also called *backward problem*. In (4.14) we see the reason for Definition 4.2. Instead of the matrix $\mathbf{A}$ in Burgers' equation (4.2), we use the matrix $\mathbf{A}^\top$ in (4.14), which is the adjoint of a real matrix.

The power of the adjoint method is illustrated in Figure 4.3. Plot 4.3a represents the normalized negative gradient of the full model obtained via centered finite differences. Plot 4.3b the normalized negative gradient obtained via the adjoint method. As seen in the derivation, the adjoint method yields the exact gradient, not taking numerical errors into account. However, it requires only the solution of one linear system, in contrast to numerous solves of a nonlinear system for the finite differences approach. Indeed, in the employed



(a) Centered Differences (b) Adjoint Method

Figure 4.3: Negative normalized gradient based on the full system for Example 2

setting ($N = 50, M = 80, \nu = 1/100$), the required time for the adjoint method is more than 700 times faster as for the centered finite differences.

Note that the adjoint systems requires the solution of the state equation. Since the adjoint equation is solved backwards in time, the whole solution of the state equation is required to determine the adjoint variables. So far, we were only concerned about the computational time. For large systems, which naturally arise in the flow control setting, it might be even impossible to store the whole solution of the forward problem. To this end, *checkpointing strategies* are employed. Assume the checkpoints $\{T_k\}_{k=1}^K$, where

$$0 \leq T_1 < T_2 < \cdots < T_{K-1} < T_K \leq T,$$

with $K \ll M$. The solution of the forward solve is only stored at the times T_k. To solve the adjoint system, different approaches are available. An inexpensive, but also mathematically not necessarily reasonable choice, is to use simply the solution of the "closest" checkpoint or employ a convex combination of the surrounding ones, which is an linear interpolation. Another option is to resolve the state equation in $[T_{k-1}, T_k]$ for $k = K, \ldots, 2$ with the solution at T_{k-1} as initial condition. This results in the correct solutions but requires an additional solve of the state equation. In a third approach we solve the state equation in the interval $[T_k, T_{k+1}]$ and directly solve the corresponding adjoint equation. Then, the next interval is considered. However, this yields discontinuities in the adjoint variables. Note, that the use of a reduced-order model might help here as well. Beside the faster computation, it has lower storage costs and checkpointing strategies might become obsolete.

The functions in terms of the POD reduced model are given as

$$\begin{aligned} e_1\left(t, \alpha_r, \dot{\alpha}_r, u\right) &= \dot{\alpha}_r(t) - \mathbf{A}_r\alpha_r(t) - \mathbf{V}^\top \mathcal{N}\left(\mathbf{V}\alpha_r(t)\right) - \mathbf{B}_r\overline{u}(t), \\ e_2\left(\alpha_r(0), u\right) &= \alpha_r(0) - \overline{\alpha}_r \end{aligned}$$

with partial derivatives

$$\begin{aligned} \partial_y e_1 &= -\mathbf{A}_r - \mathbf{V}^\top \nabla_y \mathcal{N}\left(\mathbf{V}\alpha_r(t)\right)\mathbf{V}, & \partial_{y(0)} e_2 &= \mathbf{I}_N, \\ \partial_{\dot{y}} e_1 &= \mathbf{I}_N, & \partial_u e_2 &= \mathbf{0}_N. \\ \partial_u e_1 &= -\mathbf{B}_r. \end{aligned}$$

The cost function g is $g(t, y, u) = \mathcal{J}(\mathbf{V}\alpha_r, u)$ and the partial derivatives are given by

$$\frac{\partial}{\partial u}\mathcal{J} = \beta u^\top \qquad \text{and} \qquad \frac{\partial}{\partial y}\mathcal{J} = \left(y - y^d\right)^\top \mathbf{V}.$$

Henceforth, the adjoint equation (4.12) reads as

$$\begin{cases} \dot{\lambda} = \mathbf{A}_r^\top \lambda + \left(\mathbf{V}^\top \nabla \mathcal{N}\left(\mathbf{V}\alpha(t)\right)\mathbf{V}\right)^\top \lambda - \mathbf{V}^\top\left(y - y^d\right) \\ \lambda(T) = 0 \end{cases} \tag{4.15}$$

For the POD-DEIM reduced model, the following functions and derivatives are different:

$$\begin{aligned} e_1\left(t, \alpha_r, \dot{\alpha}_r, u\right) &= \dot{\alpha}_r(t) - \mathbf{A}_r\alpha_r(t) - \mathbf{V}^\top\mathbf{U}\left(\mathbf{P}^\top\mathbf{U}\right)^{-1}\mathcal{N}\left(\tilde{\mathbf{V}}\alpha_r(t)\right) - \mathbf{B}_r\overline{u}(t), \\ \partial_y e_1 &= -\mathbf{A}_r - \mathbf{V}^\top\mathbf{U}\left(\mathbf{P}^\top\mathbf{U}\right)^{-1}\nabla_y\mathcal{N}\left(\tilde{\mathbf{V}}\alpha_r(t)\right)\tilde{\mathbf{V}} \end{aligned}$$

Accordingly, the adjoint equation reads as

$$\begin{cases} \dot{\lambda} = \mathbf{A}_r^\top \lambda + \left(\mathbf{V}^\top\mathbf{U}\left(\mathbf{P}^\top\mathbf{U}\right)^{-1}\nabla_y\mathcal{N}\left(\tilde{\mathbf{V}}\alpha_r(t)\right)\tilde{\mathbf{V}}\right)^\top \lambda - \mathbf{V}^\top\left(y - y^d\right) \\ \lambda(T) = 0 \end{cases} \tag{4.16}$$

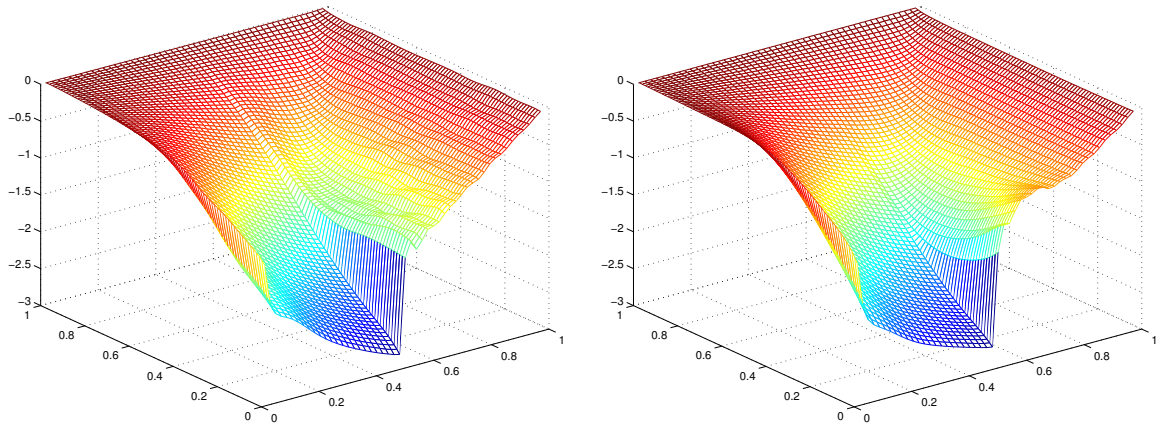
The gradients obtained via the adjoint method based on the reduced models are presented in Figure 4.4. Summarizing the previous sections yields a clearly defined routine of the TRPOD/TRPOD-DEIM algorithm. The corresponding flow chart is presented in Figure 4.5.

4.5 Line Search Algorithm for the Step Size

Recall the iteration in each step of the optimization algorithm. Assume, the function to be minimized is $g : \mathbb{R}^n \to \mathbb{R}$. First, a descent direction p_k is computed, which in our case is given by the negative gradient, that is $p_k = -\nabla f(x_k)$. Secondly, the function is optimized in the descent direction

$$\min G_k(s) = g(x_k + sp_k).$$

The minimizing argument is the step size s_k to use in the optimization. Typically, it is expensive to find a global minimum even in this one dimensional setting. An intuitive approach is to find s_k such that $G_k(s_k) < G_k(0)$. This condition is satisfied for small s_k. However, simple examples show that this might lead to very slow convergence and moreover,



(a) Centered Differences (b) Adjoint Method

Figure 4.4: Negative normalized gradient based on the POD reduced model for Example 2

the solution might not be a minimizer. Henceforth, a sufficient decrease in the objective function g is required and measured by the inequality

$$g(x_k + s_k p_k) \leq g(x_k) - c_1 s_k \|p_k\|^2, \tag{4.17}$$

for given $c_1 \in (0,1)$. The inequality (4.17) is known as *Armijo Rule* and based on [6]. Since a local minimum is achieved for a vanishing gradient, a reduction of the slope is required as well and is given by the *Curvature Condition*

$$\nabla g(x_k + s_k p_k)^\top p_k \geq c_2 \|p_k\|, \qquad c_2 \in (c_1, 1). \tag{4.18}$$

Together, the inequalities (4.17) and (4.18) are known as *Wolfe Conditions*, which go back on the work of Wolfe [71], [72]. However, (4.18) requires additional evaluations of the gradient and therefore additional solves of the adjoint equation. In order to avoid this, we apply the following rule instead. As before, let δ_k denote the trust region radius. For $0 < c_1 < c_2$, $0 < \nu_1 < 1$, and $\nu_2 > 0$, find s_k such that

$$G_k(\lambda) \leq g(x_k) - c_1 s_k \|p_k\|^2 \tag{4.19}$$

$$|s_k| \|p_k\| \leq \delta_k \tag{4.20}$$

and

$$s_k \geq \eta \qquad \text{or} \qquad s_k \geq \min\{\nu_1 \frac{\delta_k}{\|p_k\|}, \nu_2\}, \tag{4.21}$$

where $\eta > 0$ must satisfy

$$G_k(\eta) \geq g(x_k) - c_2 \eta \|p_k\|^2. \tag{4.22}$$

This is a simplified version of the rule presented in [5]. A proof that such s_k exists is given in [67].

Again, (4.19) forces a sufficient descent. The inequality (4.20) enforces to stay within the trust region radius and finally, (4.22) requires are 'large enough' step. Condition (4.22) can

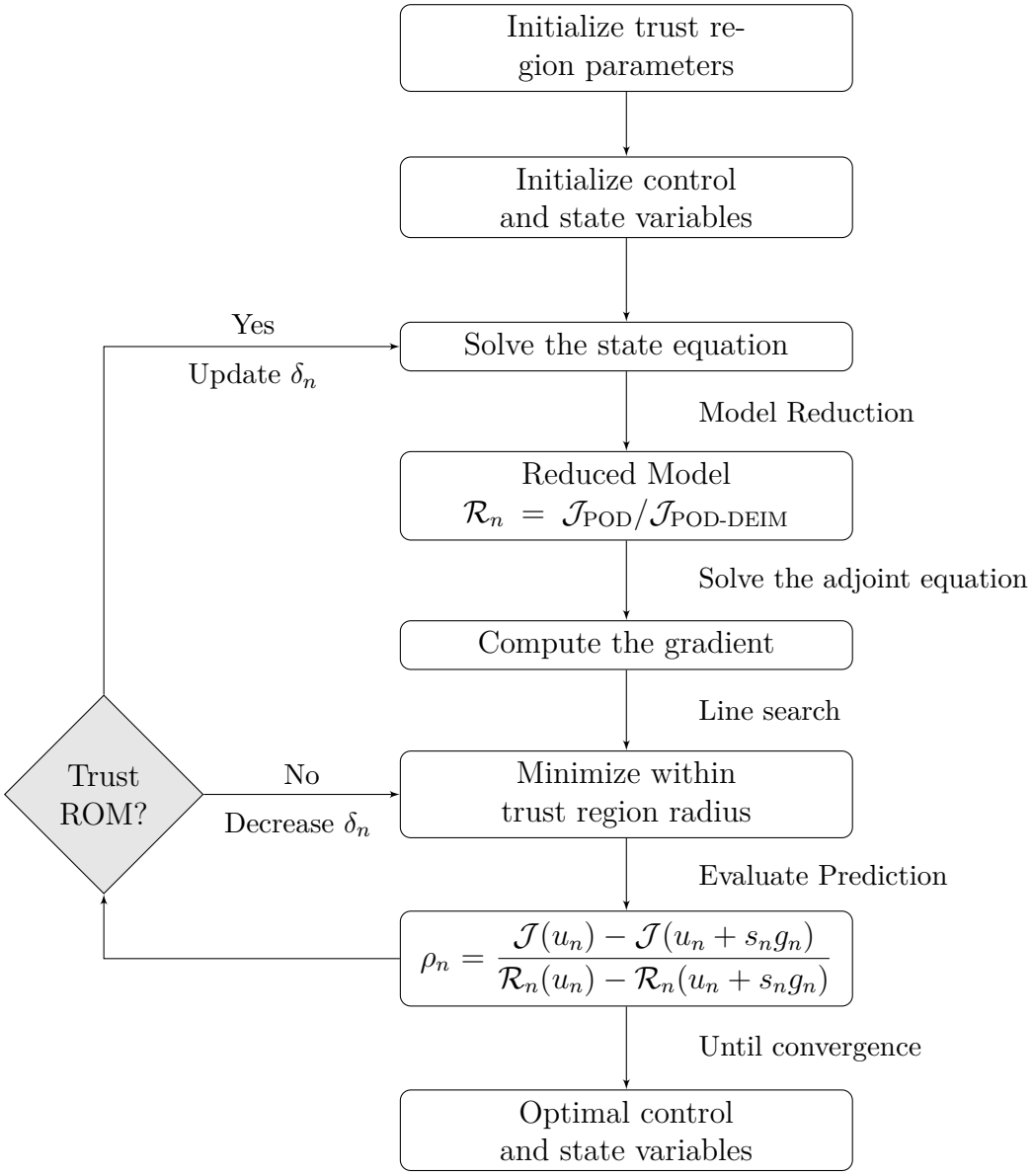


Figure 4.5: Flow chart of the TRPOD/TRPOD-DEIM algorithm

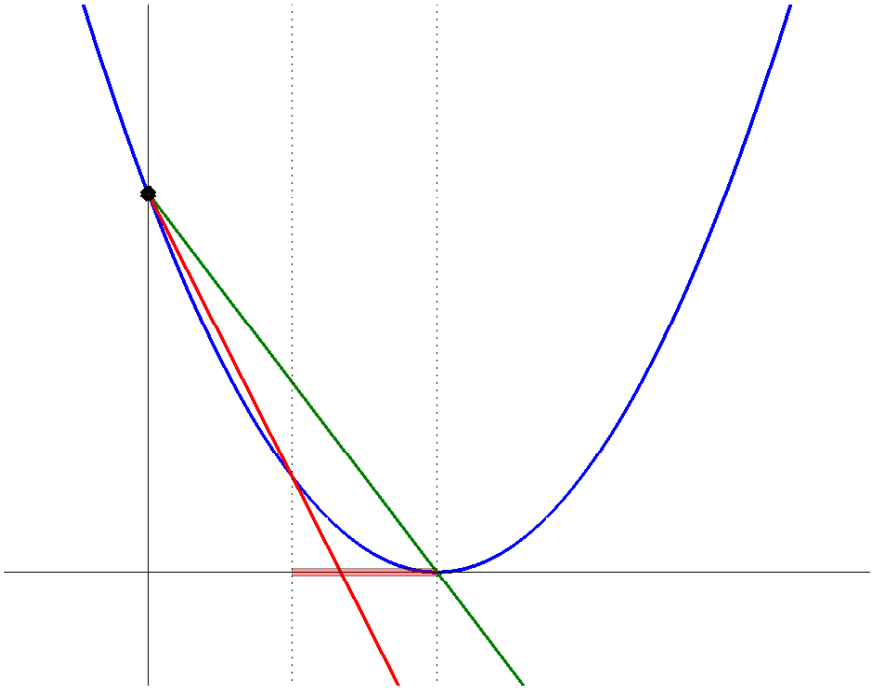


Figure 4.6: Graphical illustration of Algorithm 8

be seen as the analogue of the curvature condition (4.18). The relationship between (4.19) and (4.22) is illustrated in Figure 4.6. The blue line represents the objective function g, the green line the right hand side of (4.19) and the red line the right hand side of (4.22). The red bar indicates the area where both criterions, (4.19) and (4.22) are satisfied. Algorithm 8 guarantees, that s_k satisfies (4.19)-(4.22).

4.6 Numerical Results

We adjust the parameter setup according to Table 4.1. This table contains also the trust-region parameters. The initial condition is still the shock condition used in the previous section. All time measurements are averaged over ten repetitions. For our studies, we use a

N	M	T	ν	α	η_1	η_2	γ_1	γ_2	γ_3	γ_4	δ_0
30	50	1	0.01	0.001	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	2	0.95	2

Table 4.1: Parameter setup

fixed number of POD basis modes and DEIM interpolation points:

POD modes: $\ell = 5$, DEIM interpolation points: $k = 5$, desired state: $y^d \equiv 0$.

First, we indicate that indeed the TRPOD converges in this example. Some plots, outlining the evolution of the control, the full system and the gradient, are presented in Figure 4.7.

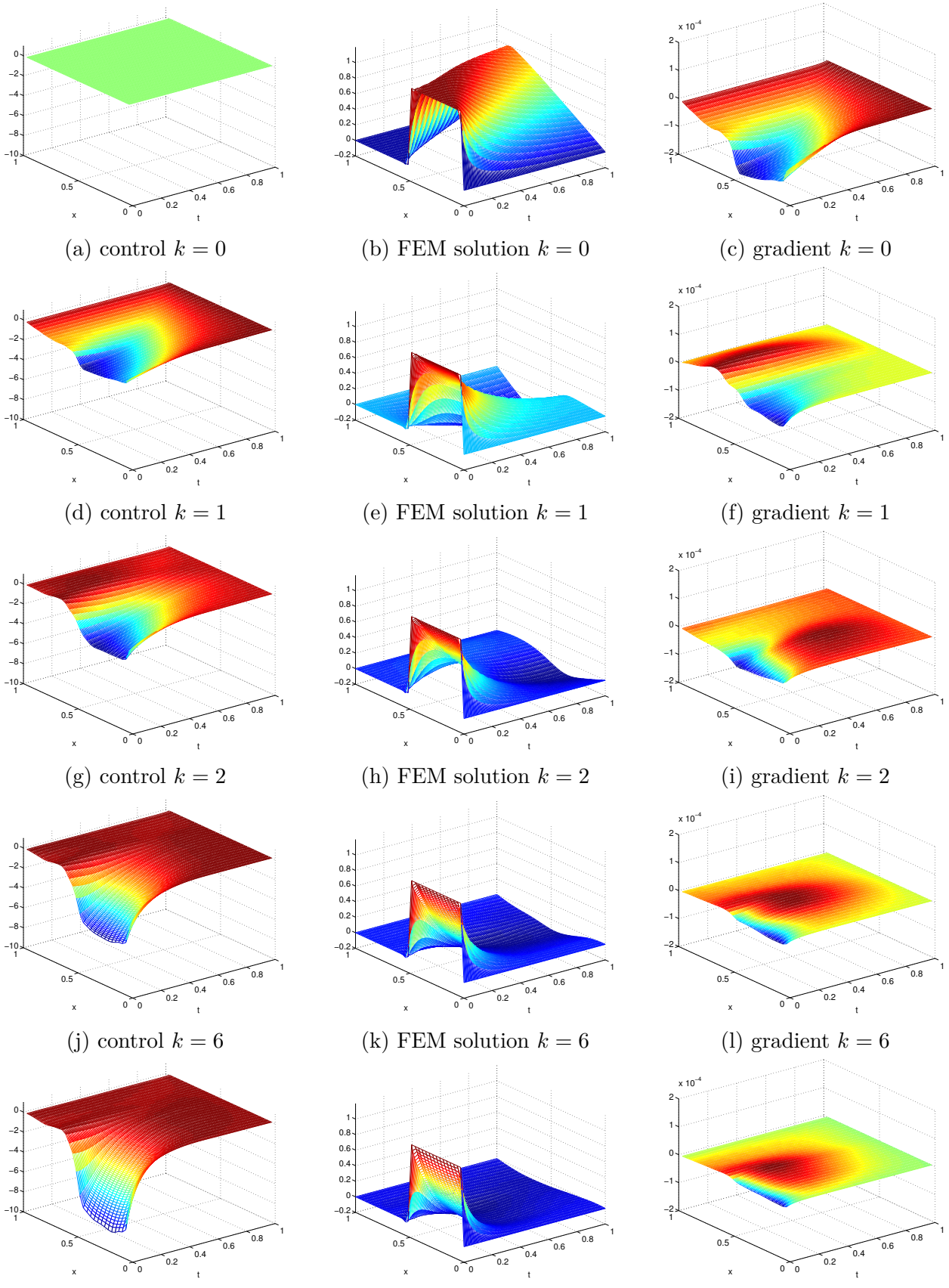
Algorithm 8 Step Size Control

Input: trust region radius δ_k, search direction p_k and constants $0 < c_1 < c_2 < 1$, $0 < \nu_1 < 1$, $\nu_2 > 0$.

Output: step size s_k.

1. Compute the norm of the search direction $ng = \|p_k\|$ and normalize the search direction $p_{k,n} = \frac{p_k}{ng}$.
2. Set $\lambda_0 = \delta_k, i = 0$. Until convergence
   (a) Compute $G_k(\lambda_i)$. If (4.19) and (4.22) are satisfied for λ. Stop!
   (b) If (4.19) is not satisfied, set $\lambda_{i+1} = \frac{1}{2}\lambda_i$.
   (c) If (4.22) is not satisfied, set $\lambda_{i+1} = \frac{1}{2}(\lambda_i + \lambda_{i-1})$.
3. Set $s_k = \lambda_i$.

The final state and final control of both algorithms are presented in Figure 4.8. In case of the TRPOD-DEIM, the FEM approximation is decaying faster to zero. This is also confirmed by the steeper magnitude of the control.



(a) control $k = 0$ (b) FEM solution $k = 0$ (c) gradient $k = 0$

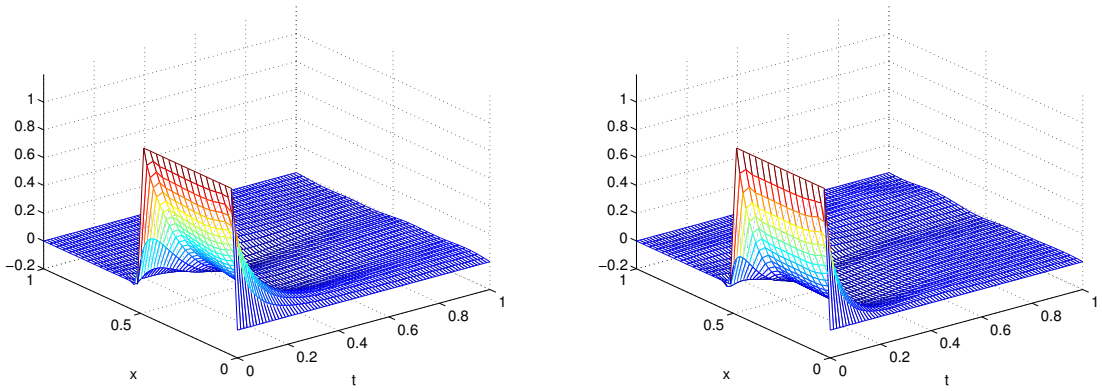
(d) control $k = 1$ (e) FEM solution $k = 1$ (f) gradient $k = 1$

(g) control $k = 2$ (h) FEM solution $k = 2$ (i) gradient $k = 2$

(j) control $k = 6$ (k) FEM solution $k = 6$ (l) gradient $k = 6$

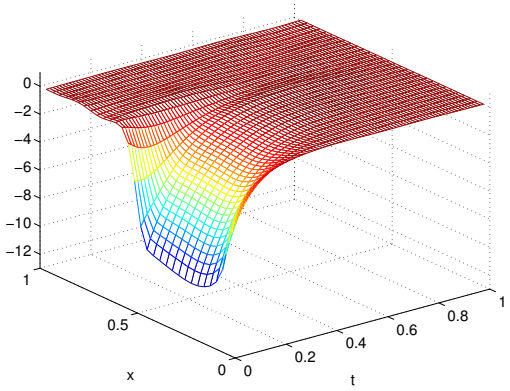
(m) control $k = 10$ (n) FEM solution $k = 10$ (o) gradient $k = 10$

Figure 4.7: Some Plots of the TRPOD iteration with $\alpha = 0.001$

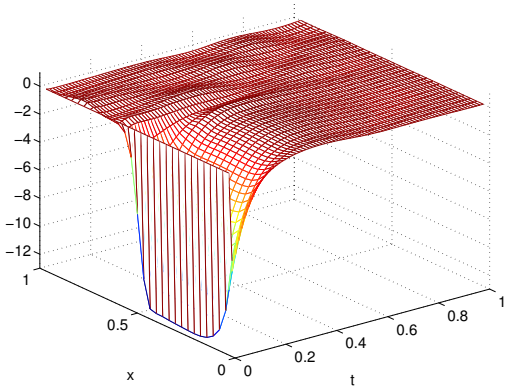


(a) Final state TRPOD

(b) Final state TRPOD-DEIM



(c) Final control TRPOD



(d) Final control TRPOD-DEIM

Figure 4.8: Final state and control of TRPOD and TRPOD-DEIM

Chapter 5

Results and Conclusions

5.1 Overview of Results

In this thesis, we studied the impact of discretization techniques on Reduced Order Models (ROMs), namely Proper Orthogonal Decomposition (POD) and Discrete Empirical Interpolation (DEIM). Basis for the conducted study is the nonlinear PDE Burgers' equation. The employed discretization techniques are a Finite Differences (FD) approach, a Finite Element Method (FEM) and a Group Finite Element Method (GFE). The resulting ODEs were integrated with a backward Euler and several MATLAB® solvers that reflect the stiffness property of Burgers' equation. All three techniques provide good numerical approximations. The GFE outperforms both other methods in terms of computational effort. The approximations of the techniques were verified using the Method of Manufactured Solutions (MMS) and a PDE solver of MATLAB® . We observed convergence over the relevant levels of discretization.

Next, we analyzed the structure of the POD basis. The POD basis for the one-way wave equation was proven to coincide with the Fourier basis. A Fourier basis like behavior appeard in the POD modes for Burgers' equation. In fact, the solution of Burgers' equation with a shock as initial condition acts as a falling wave with similar properties to the one-way wave equation. The result was used to predict the behavior of the error of the POD approximation. The second result corresponding to the POD modes was based on a decomposition of the solution of Burgers' equation into its time average and a small remaining term. The first POD modes show good agreement with the time average. Therefore, an alternative method to determine the POD basis was introduced. The new method is neither stable nor robust but yields a better understanding of the POD basis.

We presented a POD model reduction scheme to overcome the computationally expensive discretization techniques. Besides the common POD approach, a Group Proper Orthogonal Decomposition (GPOD) was employed and tested for stability and feasibility. The ROMs with POD based on FD and FEM were compared with the GPOD based system. In order to fulfill the requirement for a high fidelity of the approximation we utilized the MATLAB® solver ODE45. To avoid effects based on adaptive grids, a further study with a semi-implicit Euler was conducted. Generally, the ROMs produce good approximations with errors decaying to

zero. The POD (FD) based model showed computational advantages over the other reduced models.

In addition to POD, the nonlinear term was reduced separately with DEIM. This additional reduction entailed a further reduction of the computational time. In contrast to POD (FD/FEM) and GPOD, the GPOD-DEIM outperformed the POD-DEIM approaches.

Finally, the ROMs were studied in their application in optimal control theory. The ROMs were employed in the expensive computation of a descent direction and a line search algorithm. Hereby, computational advantages have been achieved demonstrating the effectiveness of ROMs in the optimal control setting. Special emphasis was given to the computation of the gradient. The gradient was used as descent direction in the Trust Region Framework proposed by Arian, Fahl and Sachs [5].

5.2 Conclusions

The motivation for this research was to determine whether the discretization technique has an impact on the reduced model. So far, the reduction of a system has been perceived as a process independent of the discretization technique. As outlined in the introduction, Burgers' equation is a simple nonlinear model that comprises important properties of fluid flows. Since it is proven to be an approximation of the one-dimensional unsteady Navier-Stokes equation, Burgers' equation serves as reasonable model in this thesis.

FD and FEM were analyzed with regard to reduced order modeling. They results in different reduced systems, which generate almost the same results. They vary in the required computational time. However, there is no clear advice on which technique to employ. However, a good knowledge about the discretization technique might yield a better choice of the reduction technique.

5.3 Open Problems

For the discretization with the FEM we have employed piecewise continuous basis functions. For Burgers' equation, this is a reasonable choice. In constrast, higher order PDEs require additional smoothness. Besides such smoother basis functions, Discontinuous Galerkin Methods and Wavelets are subject to recent studies and their impact on model reduction would be interesting to study.

In this thesis we observed varying stiffness properties based on the employed numerical schemes. To our knowledge, no results about the influence of POD on the stiffness of a system are available. A numerical investigation of the stiffness factor, given by the quotient of the largest and the smalles eigenvalue of the Jacobian of a system, might give a first insight.

Moreover, the application of ROMs in optimal control settings raise several research opportunities:

- May extensions of POD, for example Global POD, yield better results in the trust region setting?
- The gradient of the system was used to determine a descendent direction. Is it possible to incorporate gradient information into the POD basis in order to improve convergence?
- Analyze the impact of ROMs on the discretize-then-differentiate vs. differentiate-then-discretize discussion.
- Employ ROMs to avoid checkpointing strategies.

Bibliography

[1] K. Afanasiev and M. Hinze. Adaptive control of a wake flow using proper orthogonal decomposition, 1999.

[2] A. Agarwal and L. Biegler. A trust-region framework for constrained optimization using reduced order modeling. *Optimization and Engineering*, 14(1):3–35, 2013.

[3] A. Antoulas. *Approximation of Large-Scale Dynamical Systems*. Advances in Design And Control. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2005.

[4] A. Antoulas and D. Sorensen. Approximation of large-scale dynamical systems: An overview, 2001.

[5] E. Arian, M. Fahl, and E. W. Sachs. Trust-region proper orthogonal decomposition for flow control, 2000.

[6] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[7] J. Atwell and B. King. Proper orthogonal decomposition for reduced basis feedback controllers for parabolic equations. *Mathematical and Computer Modelling*, 33(1–3):1 – 19, 2001.

[8] M. Barrault, N. C. Nguyen, Y. Maday, and A. T. Patera. An "Empirical Interpolation" Method: Application to Efficient Reduced-Basis Discretization of Partial Differential Equations. *C. R. Acad. Sci. Paris, Série I.*, 339:667–672, 2004.

[9] M. Bergounioux, K. Ito, and K. Kunisch. Primal-dual strategy for constrained optimal control problems, 1997.

[10] M. Bergounioux and K. Kunisch. Primal-dual strategy for state-constrained optimal control problems. *Comput. Optim. Appl.*, 22(2):193–224, July 2002.

[11] G. Berkooz, P. Holmes, and J. L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Rev. Fluid Mech*, pages 539–575, 1993.

[12] J. Borggaard, A. Hay, and D. Pelletier. Interval-based reduced-order models for unsteady fluid flow, 2007.

[13] A. M. Bradley. Pde-constrained optimization and the adjoint method. online, Stanford University, 2013. http://www.stanford.edu/~ambrad/adjoint_tutorial.pdf; visited on August 7th 2013.

[14] A. Bryson and W. Denham Jr. A steepest ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 84(3):247–257, 1962.

[15] J. Burns and E. Cliff. Methods for approximating solutions to linear hereditary quadratic optimal control problems. *Automatic Control, IEEE Transactions on*, 23(1):21–36, 1978.

[16] C. I. Byrnes, D. S. Gilliam, and V. I. Shubov. On the global dynamics of a controlled viscous burgers' equation. *Journal of Dynamical and Control Systems*, 4(4):457–519, Oct. 1998.

[17] S. Chaturantabut. Dimension reduction for unsteady nonlinear partial differential equations via empirical interpolation methods. master thesis, Rice University, 2008. Available online at http://www.caam.rice.edu/tech_reports/2009/TR09-36.pdf; visited on Mai 14th 2013.

[18] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[19] S. Chaturantabut and D. C. Sorensen. Application of pod and deim on dimension reduction of non-linear miscible viscous fingering in porous media. *Mathematical and Computer Modelling of Dynamical Systems*, 17(4):337–353, 2011.

[20] S. Chaturantabut and D. C. Sorensen. A state space error estimate for pod-deim nonlinear model reduction. *SIAM Journal on Numerical Analysis*, 50(1):46–63, 2012.

[21] J. D. Cole. On a Quasi-Linear Parabolic Equation Occurring in Aerodynamics. *Quart. Appl. Math.*, 9:225–236, 1951.

[22] B. T. Dickinson. Nonlinear model reduction using the group proper orthogonal decomposition method. master thesis, Oregon State University, 2007. Available online at http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/5542/BTD_Thesis.pdf; visited on May 6th 2013.

[23] B. T. Dickinson and J. R. Singler. Nonlinear model reduction using group proper orthogonal decomposition. *International Journal of Numerical Analysis and Modeling*, 7(2):356 – 372, 2010.

[24] A. C. Duffy. An introduction to gradient computation by the discrete adjoint method. Technical Report, Florida State University, 2009. http://computationalmathematics.org/topics/files/adjointtechreport.pdf; visited on September 4th 2013.

[25] B. Engquist and A. Majda. Absorbing boundary conditions for numerical simulation of waves. *Proceedings of the National Academy of Sciences*, 74(5):1765–1766, 1977.

[26] C. Fletcher. The group finite element formulation. *Computer Methods in Applied Mechanics and Engineering*, 37(2):225 – 244, 1983.

[27] S. Gugercin and A. C. Antoulas. A Survey of Model Reduction by Balanced Truncation and Some New Results. *International Journal of Control*, 77(8):748–766, 2004.

[28] M. Gunzburger. *Perspectives in Flow Control and Optimization*. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2003.

[29] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer Series in Computational Mathematics. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2006.

[30] L. Halpern and L. N. Trefethen. Wide-angle one-way wave equations. *The Journal of the Acoustical Society of America*, 84(4):1397–1404, 1988.

[31] J. H. Hasham and Y. ming XU. Review of proper orthogonal decomposition method applied to computational aeroelasticity. *CADDM*, 20(1):65–77, 2010.

[32] R. Herzog, K. Kunisch, and J. Sass. Primal-dual methods for the computation of trading regions under proportional transaction costs. *Mathematical Methods of Operations Research*, 77(1):101–130, 2013.

[33] M. Hinze and S. Volkwein. Error estimates for abstract linear-quadratic optimal control problems using proper orthogonal decomposition. *Computational Optimization and Applications*, 39(3):319–345, 2008.

[34] P. Holmes, J. Lumley, G. Berkooz, and C. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge Monographs on Mechanics. Cambridge University Press, 2012.

[35] E. Hopf. The partial differential equation ut + uux = Îijxx. *Communications on Pure and Applied Mathematics*, 3(3):201–230, 1950.

[36] P. Hovland, B. Mohammadi, and C. Bischof. Automatic differentiation of Navier-Stokes computations. In J. Borggaard, J. Burns, E. Cliff, and S. Schreck, editors, *Computational Methods for Optimal Design and Control*, pages 265–284. Birkhäuser, Boston, 1998.

[37] K. Ito and Y. Yan. Viscous scalar conservation law with nonlinear flux feedback and global attractors. *Journal of Mathematical Analysis and Applications*, 227(1):271 – 299, 1998.

[38] C. Jarvis. Reduced order model study of burgers' equation using proper orthogonal decomposition. master thesis, Virginia Polytechnic Institute and State University, 2012. Available online at `http://www.math.vt.edu/people/chjarvis/index_files/Jarvis_CH_T_2012.pdf`; visited on April 25th 2013.

[39] A. Jüngel. Modeling and numerical approximation of traffic flows. online, lecture notes, 2002. `http://asc.tuwien.ac.at/~juengel/scripts/trafficflow.pdf`; visited on September 4th 2013.

[40] M. K. Kadalbajoo and A. Awasthi. A numerical method based on crank-nicolson scheme for burgers' equation. *Applied Mathematics and Computation*, 182(2):1430–1442, 2006.

[41] C. Kelley. *Iterative methods for optimization*. Frontiers in applied mathematics. Society for Industrial and Applied Mathematics, 1999.

[42] C. T. Kelley and E. W. Sachs. Solution of optimal control problems by a pointwise projected newton method. *SIAM J. Control Optim*, 33:1731–1757, 1995.

[43] B. Krämer. Model reduction of the coupled burgers equation in conservation form. master thesis, Virginia Polytechnic Institute and State University, 2011. Available online at `http://scholar.lib.vt.edu/theses/available/etd-08262011-115636/unrestricted/Kramer_B_T_2011.pdf`; visited on April 24th 2013.

[44] K. Kunisch and S. Volkwein. Control of the burgers equation by a reduced-order approach using proper orthogonal decomposition. *Journal of Optimization Theory and Applications*, 102(2):345–371, 1999.

[45] L. Lasdon, S. Mitter, and A. Waren. The conjugate gradient method for optimal control problems. *Automatic Control, IEEE Transactions on*, 12(2):132–138, 1967.

[46] L. Lasdon, A. Waren, and R. Rice. An interior penalty method for inequality constrained optimal control problems. *Automatic Control, IEEE Transactions on*, 12(4):388–395, 1967.

[47] M. Lighthill. Viscosity effects in sound waves of finite amplitude. In *Surveys in mechanics*, pages 250–351. Cambridge, at the University Press, 1956.

[48] M. Liu, W. Cao, and Z. Fan. Convergence and stability of the semi-implicit euler method for a linear stochastic differential delay equation. *Journal of Computational and Applied Mathematics*, 170(2):255 – 268, 2004.

[49] J. Marchal and P. Cervenka. Analysis of the burgers equation applied to parametric transmission: Influence of the phase of the primary waves and sub-harmonics generation. *Acta Acustica united with Acustica*, 90(1):410–418, 2004.

[50] R. Marchand and D. Davidson. The method of manufactured solutions for the verification of computational electromagnetics. In *Electromagnetics in Advanced Applications (ICEAA), 2011 International Conference on*, pages 487–490, 2011.

[51] B. Mohammadi. Shape optimization for 3d turbulent flows using automatic differentiation. *International Journal of Computational Fluid Dynamics*, 11(1-2):27–50, 1998.

[52] V. Q. Nguyen. A numerical study of burgers' equation with robin boundary conditions. master thesis, Virginia Polytechnic Institute and State University, 2001. Available online at `http://scholar.lib.vt.edu/theses/available/etd-02202001-143923/unrestricted/etd.pdf`; visited on April 24th 2013.

[53] R. Padhi and S. Balakrishnan. Proper orthogonal decomposition based feedback optimal control synthesis of distributed parameter systems using neural networks. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 4389–4394, 2002.

[54] B. Pagurek and C. Woodside. The conjugate gradient method for optimal control problems with bounded control variables. *Automatica*, 4(5 - 6):337 – 349, 1968.

[55] K. Pandey and L. Verma. A note on crank-nicolson scheme for burgers' equation. *Applied Mathematics*, (2):883–889, 2011.

[56] S. Pitchaiah and A. Armaou. Output feedback control of distributed parameter systems using adaptive proper orthogonal decomposition. *Industrial & Engineering Chemistry Research*, 49(21):10496–10509, 2010.

[57] S. M. Pugh. Finite element approximations of burgers' equation. master thesis, Virginia Polytechnic Institute and State University, 1995. Available online at `http://scholar.lib.vt.edu/theses/available/etd-7697-194740/unrestricted/THESIS.PDF`; visited on March 1st 2013.

[58] P. J. Roache. *Verification and validation in computational science and engineering*. Hermosa Publishers, 1998.

[59] P. J. Roache. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering*, 124(1):4–10, 2002.

[60] P. J. Roache. Building pde codes to be verifiable and validatable. *Computing in Science and Engg.*, 6(5):30–38, Sept. 2004.

[61] C. W. Rowley. Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos (IJBC)*, 15(3):997–1013, 2005.

[62] C. J. Roy, C. C. Nelson, T. M. Smith, and C. C. Ober. Verification of Euler/Navier-Stokes codes using the method of manufactured solutions. *International Journal for Numerical Methods in Fluids*, 44(6), Feb. 2004.

[63] L. F. Shampine and M. W. Reichelt. The matlab ode suite, 1997.

[64] L. Sirovich. Turbulence and the dynamics of coherent structures. i - coherent structures. ii - symmetries and transformations. iii - dynamics and scaling. *Quarterly of Applied Mathematics*, 45:561–571, 1987.

[65] L. C. Smith. Finite element approximation of burgers' equation with robin's boundary conditions. master thesis, Virginia Polytechnic Institute and State University, 1997. Available online at `http://scholar.lib.vt.edu/theses/available/etd-7697-194740/unrestricted/THESIS.PDF`; visited on April 24th 2013.

[66] R. Spigler and M. Vianello. Convergence analysis of the semi-implicit euler method for abstract evolution equations. *Numerical Functional Analysis and Optimization*, 16(5-6):785–803, 1995.

[67] P. L. Toint. Global convergence of a a of trust-region methods for nonconvex minimization in hilbert space. *IMA Journal of Numerical Analysis*, 8(2):231–252, 1988.

[68] S. Volkwein. Model reduction using proper orthogonal decomposition. lecture notes, 2011. Available online at `http://www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/POD-Vorlesung.pdf`; visited on February 27th 2013.

[69] Z. Wang, I. Akhtar, J. Borggaard, and T. Iliescu. Two-level discretizations of nonlinear closure models for proper orthogonal decomposition. *Journal of Computational Physics*, 230(1):126 – 146, 2011.

[70] Z. Wang, I. Akhtar, J. Borggaard, and T. Iliescu. Proper orthogonal decomposition closure models for turbulent flows: A numerical comparison. *Computer Methods in Applied Mechanics and Engineering*, 237 - 240(0):10–26, 2012.

[71] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):pp. 226–235, 1969.

[72] P. Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM Review*, 13(2):pp. 185–188, 1971.

Appendix A

FEM and GFEM matrices

This chapter provides the calculations of the Finite Element and the Group Finite Element Matrices and Functions defined in Section 2.2 and Section 2.3. Let us quickly recall the linear basis functions defined in these sections:

$$\phi_0(x) = \begin{cases} -\frac{1}{h}(x - x_1), & \text{for } x_0 \leq x \leq, x_1 \\ 0, & \text{otherwise,} \end{cases}$$

$$\phi_i(x) = \begin{cases} \frac{1}{h}(x - x_{i-1}), & \text{for } x_{i-1} \leq x \leq x_i, \\ -\frac{1}{h}(x - x_{i+1}), & \text{for } x_i \leq x \leq x_{i+1}, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, \ldots, N$$

$$\phi_{N+1}(x) = \begin{cases} \frac{1}{h}(x - x_N), & \text{for } x_N \leq x \leq, x_{N+1} \\ 0, & \text{otherwise.} \end{cases}$$

Mass matrix We start with the mass matrix $M_{\text{FEM}} = \left(m_{i,j}^{\text{FEM}}\right)$ of the Finite Element Method, where $m_{i,j}^{\text{FEM}} = \int_0^1 \phi_i(x)\phi_j(x)dx$. First we observe from the definition of the basis functions that

$$m_{i,j}^{\text{FEM}} = 0 \qquad \text{if } |i - j| > 1.$$

Furthermore from $m_{i,j}^{\text{FEM}} = \int_0^1 \phi_i(x)\phi_j(x)dx = \int_0^1 \phi_j(x)\phi_i(x)dx = m_{j,i}^{\text{FEM}}$, we infer that M_{FEM} is symmetric. Hence, we only need to compute the diagonal and superdiagonal entries of the matrix:

$$\begin{aligned} m_{i,j}^{\text{FEM}} &= \int_0^1 \phi_i(x)^2 dx = \int_{x_{i-1}}^{x_i} \frac{1}{h^2}(x - x_{i-1})^2 dx + \int_{x_i}^{x_{i+1}} \frac{1}{h^2}(x - x_{i+1})^2 dx \\ &= \frac{1}{h^2}\left[\frac{1}{3}x^3 - x^2 x_{i-1} + x_{i-1}^2 x\right]_{x=x_{i-1}}^{x_i} + \frac{1}{h^2}\left[\frac{1}{3}x^3 - x^2 x_{i+1} + x_{i+1}^2 x\right]_{x=x_i}^{x_{i+1}} \\ &= \frac{1}{3h^2}(\underbrace{x_i - x_{i-1}}_{=h})^3 + \frac{1}{3h^2}(\underbrace{x_{i+1} - x_i}_{=h})^3 = \frac{h}{3} \end{aligned}$$

$$\begin{aligned} m_{i,i+1}^{\text{FEM}} &= \int_0^1 \phi_i(x)\phi_{i+1}(x)dx = -\frac{1}{h^2}\int_{x_i}^{x_{i+1}} (x-x_{i+1})(x-x_1)dx \\ &= -\frac{1}{6h^2}\left[x_i^3 - 3x_i^2 x_{i+1} + 3x_i x_{i+1}^2 - x_{i+1}^3\right] = -\frac{1}{6h^2}(\underbrace{x_i - x_{i+1}}_{=-h})^3 = \frac{h}{6}. \end{aligned}$$

Summarizing the results, M_{FEM} has the form

$$M_{\text{FEM}} = \frac{h}{6}\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Stiffness matrix Next, we compute the stiffness matrix $A_{\text{FEM}} = \left(a_{i,j}^{\text{FEM}}\right)$ where

$$a_{i,j}^{\text{FEM}} = -\nu\int_0^1 \phi_i'(x)\phi_j'(x)dx.$$

Again, $A_{\text{FEM}} = A_{\text{FEM}}^T$ and we only have to compute the diagonal and superdiagonal entries. The derivative of ϕ_i is given by

$$\phi_i'(x) = \begin{cases} \frac{1}{h}, & \text{for } x_{i-1} < x < x_i, \\ -\frac{1}{h}, & \text{for } x_i < x < x_{i+1}, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1,\dots,N.$$

$$\begin{aligned} a_{i,i}^{\text{FEM}} &= -\nu\int_0^1 \phi_i'(x)^2 dx = -\nu\int_{x_{i-1}}^{x_i}\frac{1}{h^2}dx + -\nu\int_{x_i}^{x_{i+1}}\frac{1}{h^2}dx = -\nu\frac{2}{h}. \\ a_{i,i+1}^{\text{FEM}} &= -\nu\int_0^1 \phi_i'(x)\phi_{i+1}'(x)dx = \nu\int_{x_i}^{x_{i+1}}\frac{1}{h^2}dx = \nu\frac{1}{h}. \end{aligned}$$

This implies

$$A_{\text{FEM}} = \nu\frac{1}{h}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Nonlinear Term FEM To complete the FEM discussion we compute the nonlinear term $\mathcal{N}_{\text{FEM}}(\alpha(t))$. The i-th component of $\mathcal{N}_{\text{FEM}}$ is given by

$$\sum_{j=1}^{N}\sum_{k=1}^{N}\underbrace{\left(\int_0^1 \phi_j(x)\phi_k'(x)\phi_i(x)dx\right)}_{=0,\text{ if } |j-i|>1 \text{ or } |k-i|>1}\alpha_j(t)\alpha_k(t) = \sum_{j=i-1}^{i+1}\sum_{k=i-1}^{i+1}\left(\int_0^1 \phi_j(x)\phi_k'(x)\phi_i(x)dx\right)\alpha_j(t)\alpha_k(t).$$

We split up the sum and compute its components each by itself:

Case $j = k = i$:

$$\begin{aligned}\int_0^1 \phi_i(x)^2\phi_i'(x)dx &= \int_{x_{i-1}}^{x_i} \frac{1}{h^3}(x-x_{i-1})^2dx - \int_{x_i}^{x_{i+1}} \frac{1}{h^3}(x-x_{i+1})^2dx \\ &= \frac{1}{3h^3}\left((x-x_{i-1})^3\Big|_{x=x_{i-1}}^{x=x_i} - (x-x_{i+1})^3\Big|_{x=x_i}^{x=x_{i+1}}\right) \\ &= \frac{1}{3h^3}\left(\underbrace{(x_i-x_{i-1})^3}_{=h^3} + \underbrace{(x_i-x_{i+1})^3}_{=-h^3}\right) = 0.\end{aligned}$$

Case $j = i-1, k = i+1$:

$$\begin{aligned}\int_0^1 \phi_{i-1}(x)\phi_{i+1}'(x)\phi_i(x)dx &= \int_{x_{i-1}}^{x_i} \phi_{i-1}(x)\underbrace{\phi_{i+1}'(x)}_{=0}\phi_i(x)dx + \int_{x_i}^{x_{i+1}} \underbrace{\phi_{i-1}(x)}_{=0}\phi_{i+1}'(x)\phi_i(x)dx \\ &= 0.\end{aligned}$$

Case $j = i+1, k = i-1$:

$$\begin{aligned}\int_0^1 \phi_{i+1}(x)\phi_{i-1}'(x)\phi_i(x)dx &= \int_{x_{i-1}}^{x_i} \underbrace{\phi_{i+1}(x)}_{=0}\phi_{i-1}'(x)\phi_i(x)dx + \int_{x_i}^{x_{i+1}} \phi_{i+1}(x)\underbrace{\phi_{i-1}'(x)}_{=0}\phi_i(x)dx \\ &= 0.\end{aligned}$$

Case $j = k = i-1$:

$$\begin{aligned}\int_0^1 \phi_{i-1}(x)\phi_{i-1}'(x)\phi_i(x)dx &= \int_{x_{i-1}}^{x_i} \phi_{i-1}(x)\phi_{i-1}'(x)\phi_i(x)dx + \int_{x_{i1}}^{x_{i+1}} \underbrace{\phi_{i-1}(x)\phi_{i-1}'(x)}_{=0}\phi_i(x)dx \\ &= \frac{1}{h^3}\int_{x_{i-1}}^{x_i} x^2 - xx_i - xx_{i-1} + x_ix_{i-1}dx \\ &= \frac{1}{h^3}\left(\frac{1}{3}x^3 - \frac{1}{2}x^2x_i - \frac{1}{2}x^2x_{i-1} + x_ix_{i-1}x\Big|_{x=x_{i-1}}^{x=x_i}\right) \\ &= -\frac{1}{6h^3}\left(x_i^3 - 3x_i^2x_{i-1} + 3x_ix_{i-1}^2 - x_{i-1}^3\right) = -\frac{1}{6h^3}\left(x_i - x_{i-1}\right)^3 \\ &= -\frac{1}{6}.\end{aligned}$$

Case $j = k = i+1$:

$$\begin{aligned}\int_0^1 \phi_{i+1}(x)\phi_{i+1}'(x)\phi_i(x)dx &= -\int_{x_i}^{x_{i+1}} \frac{1}{h^3}(x-x_i)(x-x_{i+1})dx \\ &= -\frac{1}{h^3}\int_{x_i}^{x_{i+1}} x^2 - x(x_i+x_{i+1}) + x_ix_{i+1}dx = \frac{1}{6h^3}\left(x_{i+1}-x_i\right)^3 \\ &= \frac{1}{6}.\end{aligned}$$

Case $j = i-1, k = i$:

$$\int_0^1 \phi_{i-1}(x)\phi_i'(x)\phi_i(x)dx = -\int_{x_{i-1}}^{x_i} \frac{1}{h^3}(x-x_i)(x-x_{i-1})dx = \frac{1}{6}.$$

Case $j = i+1, k = i$:

$$\int_0^1 \phi_{i+1}(x)\phi_i'(x)\phi_i(x)dx = \int_{x_{i1}}^{x_{i+1}} \frac{1}{h^3}(x-x_i)(x-x_{i+1})dx = -\frac{1}{6}.$$

Case $j = i, k = i-1$:

$$\int_0^1 \phi_i(x)^2\phi_{i-1}'(x)dx = -\int_{x_{i-1}}^{x_i} \frac{1}{h^3}(x-x_{i-1})^2dx = -\frac{1}{3h^3}\left(x-x_{i-1}\right)^3\Big|_{x=x_{i-1}}^{x=x_i} = -\frac{1}{3}.$$

Case $j = i, k = i+1$:

$$\int_0^1 \phi_i(x)^2\phi_{i+1}'(x)dx = \int_{x_i}^{x_{i+1}} \frac{1}{h^3}(x-x_{i+1})^2dx = \frac{1}{3h^3}\left(x-x_{i+1}\right)^3\Big|_{x=x_i}^{x=x_{i+1}} = \frac{1}{3}.$$

All together yields to following expression for the i-th nonlinear term:

$$\begin{aligned}
&-\sum_{j=1}^{N}\sum_{k=1}^{N}\left(\int_0^1 \phi_j(x)\phi_k'(x)\phi_i(x)dx\right)\alpha_j(t)\alpha_k(t)\\
=&-\sum_{j=i-1}^{i+1}\sum_{k=i-1}^{i+1}\left(\int_0^1 \phi_j(x)\phi_k'(x)\phi_i(x)dx\right)\alpha_j(t)\alpha_k(t)\\
=&-\left(-\frac{1}{6}\alpha_{i-1}^2(t)+\left(\frac{1}{6}-\frac{1}{3}\right)\alpha_{i-1}\alpha_i(t)+\left(\frac{1}{3}-\frac{1}{6}\right)\alpha_i\alpha_{i+1}(t)+\frac{1}{6}\alpha_{i+1}^2(t)\right)\\
=&\frac{1}{6}\left(\alpha_{i-1}^2+\alpha_{i-1}(t)\alpha_i(t)-\alpha_i(t)\alpha_{i+1}(t)-\alpha_{i+1}^2(t)\right).
\end{aligned}$$

Recall that $\alpha_0 \equiv 0 \equiv \alpha_N$ and hence, the nonlinear term is given by

$$\mathcal{N}_{\text{FEM}}\left(\alpha(t)\right) = \frac{1}{6}\begin{pmatrix} -\alpha_1(t)\alpha_2(t)-\alpha_2^2(t)\\ \alpha_1^2(t)+\alpha_1(t)\alpha_2(t)-\alpha_2(t)\alpha_3(t)-\alpha_3^2(t)\\ \vdots\\ \alpha_{N-2}^2(t)+\alpha_{N-2}(t)\alpha_{N-1}(t)-\alpha_{N-1}(t)\alpha_N(t)-\alpha_N^2(t)\\ \alpha_{N-1}^2(t)+\alpha_{N-1}(t)\alpha_N(t)\end{pmatrix} \in \mathbb{R}^N.$$

Nonlinear term GFEM Finally, we are left with the nonlinear matrix $\mathcal{N}_{\text{GFE}} = \left(n_{i,j}^{\text{GFE}}\right)$ of the Group Finite Element Method. Here, $n_{i,j}^{\text{GFE}} = \frac{1}{2}\int_0^1 \phi_i'(x)\phi_j(x)dx$. In contrast to the previous computations, $\mathcal{N}_{\text{GFE}}$ is not symmetric so we need to compute also the subdiagonal

entries.

$$\begin{aligned}
n_{i,i}^{\mathrm{GFE}} &= \frac{1}{2}\int_0^1 \phi_i'(x)\phi_i(x)dx = \frac{1}{2}\int_{x_{i-1}}^{x_i}\frac{1}{h^2}(x-x_{i-1})dx + \frac{1}{2}\int_{x_i}^{x_{i+1}}\frac{1}{h^2}(x-x_{i+1})dx \\
&= \frac{1}{2h^2}\left[\frac{1}{2}x^2 - x_{i-1}x\right]_{x=x_{i-1}}^{x_i} + \frac{1}{2h^2}\left[\frac{1}{2}x^2 - x_{i+1}x\right] - x = {x_i}^{x_{i+1}} \\
&= \frac{1}{4h^2}\left(\underbrace{x_i - x_{i-1}}_{=h}\right)^2 - \frac{1}{4h^2}\left(\underbrace{x_{i+1} - x_i}_{=h}\right)^2 = \frac{1}{4} - \frac{1}{4} = 0 \\
n_{i,i+1}^{\mathrm{GFE}} &= \frac{1}{2}\int_0^1 \phi_i'(x)\phi_{i+1}(x)dx = -\frac{1}{2h^2}\int_{x_i}^{x_{i+1}}(x-x_i)dx = -\frac{1}{2h^2}\left[\frac{1}{2}x^2 - x_i x\right]_{x=x_{i-1}}^{x_i} \\
&= -\frac{1}{4h^2}(x_{i+1}-x_i)^2 = -\frac{1}{4} \\
n_{i,i-1}^{\mathrm{GFE}} &= \frac{1}{2}\int_0^1 \phi_i'(x)\phi_{i-1}(x)dx = -\frac{1}{2h^2}\int_{x_{i-1}}^{x_i}(x-x_{i-1})dx = -\frac{1}{2h^2}\left[\frac{1}{2}x^2 - x_{i-1}x\right]_{x=x_{i-1}}^{x_i} \\
&= \frac{1}{4h^2}(x_i - x_{i-1})^2 = \frac{1}{4}.
\end{aligned}$$

Hence, the matrix $\mathcal{N}_{\mathrm{GFE}}$ is given by

$$\mathcal{N}_{\mathrm{GFE}} = \frac{1}{4}\begin{pmatrix} 0 & -1 & & & \\ 1 & 0 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & -1 \\ & & & 1 & 0 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Appendix B

Matlab® code

This chapter provides some of the employed MATLAB® code. Due to the detailed description of the algorithms and implementation issues, only exemplary code is shown.

The function to compute the GFE approximation to Burgers' equation.

```
function [sol,t,time,M,nonlinSNAP] = burgersGFE(nu,initial,f,domain,N,m,solver
    )
%-------------------------------------------------------------------------
%  burgersGFE - Computes the GFE solution of Burgers' equation
%
%  Copyright (c) 2013, Benjamin Unger, Virginia Tech
%  Version: 1.0
%
%  Variables:   nu      viscosity parameter (reciprocal of Reynolds number)
%               initial initial condition (reference on a function)
%               f       forcing term (reference on a function or a vector)
%               domain  = [x0 x1 T]
%               N       # of spatial discretization points
%               M       # of time discretization points
%               solver  specifies the solver (bEuler,ode45,ode23)
%
%  Global variables:
%               adaptiveGrid    = 0 (uses the adaptive grid for the
%                                   solution of the matlab solvers)
%                               = 1 (uses the grid defined by m and the
%                                   domain)
%
%  Supporting functions:
%       - burgersODE    ODE implementation of the discretized Burgers'
%                       equation for the matlab solver
%-------------------------------------------------------------------------
    global adaptiveGrid;
%-------------------------------------------------------------------------
%  Define Parameters
%-------------------------------------------------------------------------
    x0      = domain(1);
    x1      = domain(2);
```

```
32      T       = domain(3);
33
34      h       = (x1-x0)/(N+1);    % spatial stepsize
35      dT      = T/(m-1);          % time stepsize
36
37      err     = 10^-6;            % error for Newton's method
38      maxIt   = 100;              % maximal iterations for Newton's method
39
40      x       = linspace(x0,x1,N+2);
41      xinner  = x(2:end-1);
42      t = linspace(0,T,m);
43  %-------------------------------------------------------------------------
44  %  Initialize Discretization Matrices
45  %-------------------------------------------------------------------------
46      e       = ones(N,1);            % unit vector
47
48      M       = h/6*spdiags([e 4*e e],-1:1,N,N);
49      A       = nu/h*spdiags([e -2*e e],-1:1,N,N);
50      B       = h/4*spdiags([e 2*e e],-1:1,N,N);
51      nonlinear = -1/4*spdiags([-e 0*e e],-1:1,N,N);
52
53      options = odeset('mass',M);
54  %-------------------------------------------------------------------------
55  %  Use specified solver
56  %-------------------------------------------------------------------------
57      y0 = initial(x);      % initial condition
58      timer = tic;
59      switch solver
60          case 'bEuler'
61              sol         = zeros(N+2,m);
62              sol(:,1)    = y0;
63              y           = y0(2:end-1);
64              % work through all time steps
65              for i=2:m
66                  k = 0;
67                  % get initial condition for Newton by explicit Euler
68                  w = M*y - dT*(A*y + nonlinear*y.^2 + B*f(dT*i,xinner,nu));
69                  w = M\w;
70
71                  % compute the current G
72                  Gu = M*(w - y) - dT*(A*w + nonlinear*w.^2 + B*f(dT*i,xinner,nu
                        ));
73
74                  % use Newton to find the root
75                  while (norm(Gu,2) > err && k < maxIt)
76                      % compute the gradient of the nonlinear term
77                      GradNonlinear = 2*nonlinear*diag(w);
78
79                      % compute the current gradient
80                      GradG   = M - dT*(A + GradNonlinear);
81
82                      % solve the linear system
83                      deltaU  = -GradG\Gu;
84
```

```
                    % update
                    w       = w+deltaU;
                    Gu      = M*(w - y) - dT*(A*w + nonlinear*w.^2 + B*f(dT*i,
                        xinner,nu));
                    k       = k+1;
                end
                y = w;
                sol(2:N+1,i) = y;
            end
        otherwise
            % solver exists as a .m File or builtin function
            if exist(solver,'file') == 2 || exist(solver,'builtin') == 5
                if adaptiveGrid
                    [t,Y] = feval(solver,@burgersODE,[0 T],y0(2:end-1),options
                        ,A,B,nonlinear,f,nu,domain,N);
                else
                    [t,Y] = feval(solver,@burgersODE,t,y0(2:end-1),options,A,B
                        ,nonlinear,f,nu,domain,N);
                end
                sol = zeros(N+2,length(t));
                sol(2:end-1,:) = Y'; sol(:,1) = y0;
            else
                disp('Error: unknown solver!');
            end
    end
    time = toc(timer);

    if nargout == 5
        nonlinSNAP = nonlinear*(sol(2:end-1,:).^2);
    end
end

%-------------------------------------------------------------------------
%  Supporting Functions
%-------------------------------------------------------------------------
function xprime = burgersODE(t,x,A,B,nonlinear,f,nu,domain,N)
    spat        = linspace(domain(1),domain(2),N+2);
    xprime      = A*x + nonlinear*x.^2 + B*f(t,spat(2:end-1),nu);
end
```

The algorithm to determine the alternative POD basis.

```
function [ POD, lambda ] = altPOD(Y,M,m,l)
%-------------------------------------------------------------------------
%  altPOD - Computes the POD basis of rank l for the matrix A with Mass
%  matrix M via power method with good initial data for speedup
%
%  Copyright (c) 2013, Benjamin Unger, Virginia Tech
%  Version: 0.1
%
%  Variables:   Y       n,m data matrix
%               M       n,n mass matrix
%               m       # of grid points for time discretization
```

```
%               l       desired # of POD modes
%
%
%  Supporting functions
%   - normM             weighted inner product with mass matrix M
%   - power_method      special implementation of the power iteration
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%  Setup matrices and resulting vectors
%-------------------------------------------------------------------------
    R       = chol(full(M),'lower');
    W       = R*Y(2:end-1,2:end);
    A       = W*W';

    lambda = zeros(l,1);
    POD = zeros(length(M),l);
%-------------------------------------------------------------------------
%  Compute first POD mode
%-------------------------------------------------------------------------
    tAv = normM(1/m*sum(Y(2:end-1,:),2),M);
    [x1,x2,lambda(1)] = power_method(A,tAv);

    x2 = normM(x2,M);
    POD(:,1) = normM(x1,M);

    if norm(tAv-POD(:,1)) < norm(tAv+POD(:,1))
        err = (tAv-POD(:,1));
    else
        err = (tAv+POD(:,1));
    end
    err = normM(err,M);
%-------------------------------------------------------------------------
%  Compute other POD modes
%-------------------------------------------------------------------------
    for i=2:l
        [x,x2,lambda(i)] = power_method(A,err,err'*A*err/(err'*err));
        x2 = normM(x2,M);
        POD(:,i) = normM(x,M);

        if norm(x2-POD(:,i)) < norm(x2+POD(:,i))
            err = (x2-POD(:,i));
        else
            err = (x2+POD(:,i));
        end

        err = normM(err,M);
    end
end

function x = normM(x,M)
    x = x/sqrt(x'*M*x);
    %x = x/sqrt(x(2:end-1)'*M*x(2:end-1));
end
```

```
66
67 function [ x, x2, lambda ] = power_method( A,x,shift )
68 %-------------------------------------------------------------------------
69 %  power_method - implementation of the invers power iteration with
70 %  shifting strategy.
71 %
72 %  Copyright (c) 2013, Benjamin Unger, Virginia Tech
73 %  Version: 0.1
74 %
75 %  Variables:   A       n,n matrix
76 %               x       initial vector
77 %               shift   initial shift (optional)
78 %
79 %  Output:      x       the computed eigenvector
80 %               x2      the vector after the first iteration
81 %               lambda  corresponding eigenvalue
82 %
83 %-------------------------------------------------------------------------
84     if nargin < 3
85         shift = 0;
86     end
87 %-------------------------------------------------------------------------
88 %  define parameters
89 %-------------------------------------------------------------------------
90     k       = 0;
91     dif     = inf;
92     err     = 10^-5;
93     maxIt   = 10;
94     minIt   = 2;
95 %-------------------------------------------------------------------------
96 %  power iteration
97 %-------------------------------------------------------------------------
98     while (dif > err && k <= maxIt) || k <= minIt
99         k = k+1;
100        if shift == 0
101            z = A*x;
102        else
103            B = A-shift*eye(size(A));
104            z = B\x;
105        end
106        % update shift
107        if k > 1
108            shift   = shift + 1/z(nn);
109            dif     = 1/z(nn);
110        end
111        [~,nn] = max(abs(z));
112        x = z/z(nn);
113        if k == (minIt-1)
114           x2 = x;
115        end
116    end
117    %fprintf('# iterations: %d\n',k);
118    lambda = shift;
119 end
```

The POD (FEM) reduced model.

```
function [sol,t,time] = burgersFEM_POD(nu,initial,f,domain,N,m,solver,V)
%-------------------------------------------------------------------------
%  burgersFEM_POD - Computes the POD (FEM) approximation of the solution to
%  Burgers equation
%
%  Copyright (c) 2013, Benjamin Unger, Virginia Tech
%  Version: 1.0
%
%  Variables:   nu      viscosity parameter (reciprocal of Reynolds number)
%               initial initial condition (reference on a function)
%               f       forcing term (reference on a function or a vector)
%               domain  = [x0 x1 T]
%               N       # of spatial discretization points
%               m       # of time discretization points
%               solver  specifies the solver (bEuler,ode45,ode23)
%               V       POD basis
%
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%  Define Parameters
%-------------------------------------------------------------------------
    x0      = domain(1);
    x1      = domain(2);
    T       = domain(3);

    h       = (x1-x0)/(N+1);    % spatial stepsize
    dT      = T/(m-1);          % time stepsize

    x       = linspace(x0,x1,N+2);
    xinner  = x(2:end-1);

    % assign time variable
    t = linspace(0,T,m);
%-------------------------------------------------------------------------
%  Initialize Discretization Matrices
%-------------------------------------------------------------------------
    e           = ones(N,1);            % unit vector

    M           = h/6*spdiags([e 4*e e],-1:1,N,N);
    A           = nu/h*spdiags([e -2*e e],-1:1,N,N);
    B           = h/4*spdiags([e 2*e e],-1:1,N,N);

    nonlinear   = spdiags([e 0*e -e],-1:1,N,N);

    Vinner      = V(2:end-1,:);
    Ar          = Vinner'*A*Vinner;
    Br          = Vinner'*B;

```

```
    options = [];
%-------------------------------------------------------------------------%
%  Use specified solver
%-------------------------------------------------------------------------%
    y0 = Vinner'*M*initial(xinner);      % initial condition
    timer = tic;
    switch solver
        case 'semiEuler'
            solR        = zeros(size(V,2),m);
            solR(:,1)   = y0;
            y           = y0;

            I           = eye(size(Ar));

            % work through all time steps
            for i=2:m
                tmp     = Vinner*y;
                nlin    = 1/6*(nonlinear*(tmp.^2) + tmp.*(nonlinear*tmp));
                y = (I-dT*Ar)\(y+dT*(Vinner'*nlin + Br*f(dT*i,xinner,nu)));
                solR(:,i) = y;
            end

            % project solution
            sol = zeros(N+2,m);
            sol(2:end-1,:) = Vinner*solR;
        otherwise
            % solver exists as a .m File or builtin function
            if exist(solver,'file') == 2 || exist(solver,'builtin') == 5
                [t,Y] = feval(solver,@burgersODE,t,y0,options,Ar,Br,nonlinear,
                    Vinner,f,nu,domain,N);

                % project solution
                sol = zeros(N+2,m);
                sol(2:end-1,:) = Vinner*Y';
            else
                disp('Error: unknown solver!');
            end
    end
    % smooth out initial condition
    sol(:,1) = initial(x);

    % stop timer
    time = toc(timer);
end

function xprime = burgersODE(t,x,Ar,Br,nonlinear,V,f,nu,domain,N)
    spat        = linspace(domain(1),domain(2),N+2);
    nlin        = 1/6*(nonlinear*((V*x).^2) + (V*x).*(nonlinear*(V*x)));
    xprime      = Ar*x + V'*nlin + Br*f(t,spat(2:end-1),nu);
end
```

The ROM based on Group POD.

```
function [sol,t,time] = burgersGPOD(nu,initial,f,domain,N,m,solver,V)
%-------------------------------------------------------------------------
%  burgersGPOD - Computes the GPOD approximation of the solution to
%  Burgers equation
%
%  Copyright (c) 2013, Benjamin Unger, Virginia Tech
%  Version: 1.0
%
%  Variables:   nu      viscosity parameter (reciprocal of Reynolds number)
%               initial initial condition (reference on a function)
%               f       forcing term (reference on a function or a vector)
%               domain  = [x0 x1 T]
%               N       # of spatial discretization points
%               m       # of time discretization points
%               solver  specifies the solver (bEuler,ode45,ode23)
%               V       POD basis
%
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%  Define Parameters
%-------------------------------------------------------------------------
    x0      = domain(1);
    x1      = domain(2);
    T       = domain(3);

    h       = (x1-x0)/(N+1);    % spatial stepsize
    dT      = T/(m-1);          % time stepsize

    x       = linspace(x0,x1,N+2);
    xinner  = x(2:end-1);

    % assign time variable
    t = linspace(0,T,m);
%-------------------------------------------------------------------------
%  Initialize Discretization Matrices
%-------------------------------------------------------------------------
    e       = ones(N,1);            % unit vector

    A       = nu/h*spdiags([e -2*e e],-1:1,N,N);
    B       = h/4*spdiags([e 2*e e],-1:1,N,N);
    nonlinear = -1/4*spdiags([-e 0*e e],-1:1,N,N);

    M       = h/6*spdiags([e 4*e e],-1:1,N,N);

    Vinner  = V(2:end-1,:);
    Ar      = Vinner'*A*Vinner;
    Br      = Vinner'*B;
    nlin    = Vinner'*nonlinear;

    options = [];
%-------------------------------------------------------------------------
%  Use specified solver
```

```
%-------------------------------------------------------------------------
    y0 = Vinner'*M*initial(xinner);      % initial condition
    timer = tic;
    switch solver
        case 'semiEuler'
            solR        = zeros(size(V,2),m);
            solR(:,1)   = y0;
            y           = y0;

            I           = eye(size(Ar));

            % work through all time steps
            for i=2:m
                nlinear = nlin*diag(Vinner*y)*(Vinner*y);
                y = (I-dT*Ar)\(y+dT*(nlinear + Br*f(dT*i,xinner,nu)));
                solR(:,i) = y;
            end

            % project solution
            sol = zeros(N+2,m);
            sol(2:end-1,:) = Vinner*solR;
        otherwise
            % solver exists as a .m File or builtin function
            if exist(solver,'file') == 2 || exist(solver,'builtin') == 5
                [t,Y] = feval(solver,@burgersODE,t,y0,options,Ar,Br,nlin,
                    Vinner,f,nu,domain,N);

                % project solution
                sol = zeros(N+2,m);
                sol(2:end-1,:) = Vinner*Y';
            else
                disp('Error: unknown solver!');
            end
    end
    % smooth out initial condition
    sol(:,1) = initial(x);

    % stop timer
    time = toc(timer);
end

function xprime = burgersODE(t,x,Ar,Br,nlin,V,f,nu,domain,N)
    spat        = linspace(domain(1),domain(2),N+2);
    xprime      = Ar*x + nlin*diag(V*x)*(V*x) + Br*f(t,spat(2:end-1),nu);
end
```

The POD-DEIM approximation based on the FD model.

```
function [sol,t,time] = burgersFD_POD_DEIM(nu,initial,f,domain,N,m,solver,V,P,
    U)
%-------------------------------------------------------------------------
%  burgersFD_POD_DEIM - Integrates the POD reduced model of the Finite
%  Differences approximation of Burgers equation
```

```
5  %
6  %  Copyright (c) 2013, Benjamin Unger, Virginia Tech
7  %  Version: 1.0
8  %
9  %  Variables:   nu      viscosity parameter (reciprocal of Reynolds number)
10 %               initial initial condition (reference on a function)
11 %               f       forcing term (reference on a function or a vector)
12 %               domain  = [x0 x1 T]
13 %               N       # of spatial discretization points
14 %               M       # of time discretization points
15 %               solver  specifies the solver (bEuler,ode45,ode23)
16 %               V       projection matrix from POD
17 %               P       permutation matrix from DEIM
18 %               U       projection matrix from DEIM
19 %
20 % Supporting functions:
21 %       - burgersODE    ODE implementation of the ROM for the matlab solver
22 %       - DEIMpush      Determines the matrix P2
23 %       - DEIMsort      Sorts the employed nodes obtained via P
24 %-------------------------------------------------------------------------
25
26 %-------------------------------------------------------------------------
27 %  Define Parameters
28 %-------------------------------------------------------------------------
29     x0      = domain(1);
30     x1      = domain(2);
31     T       = domain(3);
32
33     h       = (x1-x0)/(N+1);    % spatial stepsize
34     dT      = T/(m-1);          % time stepsize
35
36     x       = linspace(x0,x1,N+2);
37     xinner  = x(2:end-1);
38
39     % assign time variable
40     t = linspace(0,T,m);
41
42     DEIMindex = size(U,2);
43 %-------------------------------------------------------------------------
44 %  Initialize Discretization Matrices
45 %-------------------------------------------------------------------------
46     e       = ones(N,1);            % unit vector
47
48     Vinner  = V(2:end-1,:);
49
50     A       = nu/h^2*spdiags([e -2*e e],-1:1,N,N);
51     Ar      = Vinner'*A*Vinner;
52     nonlinear = 1/(2*h)*spdiags([e 0*e -e],-1:1,N,N);
53
54     nlin    = P'*nonlinear;
55
56     % go over the DEIM procedure to check weather I can do it in this
57     % fashion
58     precon = (Vinner'*U)/(P'*U);
```

```
59      precon2 = P'*Vinner;
60
61      %[nlin_tmp,P2] = DEIMpush(nlin,DEIMindex,N);
62      [nlin,P2] = DEIMpush(nlin,DEIMindex,N);
63
64      %nlin    = precon*nlin_tmp;
65      Pin     = P2*Vinner;
66
67      options = [];
68  %-------------------------------------------------------------------------
69  %  Use specified solver
70  %-------------------------------------------------------------------------
71      y0 = V'*initial(x);      % initial condition
72
73      timer = tic;
74      switch solver
75          case 'semiEuler'
76              solR        = zeros(size(V,2),m);
77              solR(:,1)   = y0;
78              y           = y0;
79
80              I           = eye(size(Ar));
81
82              % work through all time steps
83              for i=2:m
84                  nonlin = precon*((precon2*y).*(nlin*(Pin*y)));
85                  %nonlin = Vinner'*((Vinner*y).*(nonlinear*(Vinner*y)));
86                  y = (I-dT*Ar)\(y+dT*(nonlin + Vinner'*f(dT*i,xinner,nu)));
87                  solR(:,i) = y;
88              end
89
90              % project solution
91              sol = V*solR;
92          otherwise
93              % solver exists as a .m File or builtin function
94              if exist(solver,'file') == 2 || exist(solver,'builtin') == 5
95                  [t,Y] = feval(solver,@burgersODE,t,y0,options,Ar,Vinner,nlin,
                        precon,precon2,Pin,f,nu,domain,N);
96                  sol = V*Y';
97              else
98                  disp('Error: unknown solver!');
99              end
100     end
101     % smooth out initial condition
102     sol(:,1) = initial(x);
103
104     % stop timer
105     time = toc(timer);
106 end
107 %-------------------------------------------------------------------------
108 %  Supporting Functions
109 %-------------------------------------------------------------------------
110 function xprime = burgersODE(t,x,Ar,V,nlin,precon,precon2,Pin,f,nu,domain,N)
111     spat        = linspace(domain(1),domain(2),N+2);
```

```
112
113     xprime      = Ar*x + precon*((precon2*x).*(nlin*(Pin*x))) + V'*f(t,spat(2:
            end-1),nu);
114 end
115
116 function [Nr,PDeim2] = DEIMpush(Nr,DEIMindex,n)
117     ind = [];
118     for i=1:DEIMindex
119         ind = [ind find(Nr(i,:))];
120     end
121
122     ind = DEIMsort(ind);
123     Nr = Nr(:,ind);
124
125     PDeim2 = zeros(length(ind),n);
126     for i=1:length(ind)
127         PDeim2(i,ind(i)) = 1;
128     end
129 end
130
131 function ind = DEIMsort(ind)
132     ind = sort(ind);
133
134     del = [];
135     for i=1:length(ind)-1
136         if ind(i) == ind(i+1)
137             del = [del i];
138         end
139     end
140
141     ind(del) = [];
142 end
```

The TRPOD algorithm.

```
1  function [u,Y,t,x,cost,costR,k,time] = burgersTRPOD( )
2  %-------------------------------------------------------------------------
3  %  burgersTRPOD  -  Optimal Control of a Burgers' equation using the TRPOD
4  %                   method proposed in the paper 'Trust-Region Proper
5  %                   Orthogonal Decomposition for Flow Control' published by
6  %                   Eyal Adrian, Marco Fahl, Ekkehard Sachs
7  %
8  %  Copyright (c) 2013, Benjamin Unger, Virginia Tech
9  %  Version: 1.0
10 %
11 %  Usage:    [] = burgersTRPOD(u0)
12 %
13 %  Variables:  nu       viscosity parameter (reciprocal of Reynolds number)
14 %              u0       initial control
15 %              y0       initial condition
16 %              yd       desired state
17 %              p        # of POD basis vectors
18 %              n        # of spatial discretization points
```

```
%               m        # of time discretization points
%               T        end point of time interval (0,T)
%               alpha    control cost measure
%
%-------------------------------------------------------------------------
    clc; close all;
%-------------------------------------------------------------------------
%  Define Parameters (to be outsourced)
%-------------------------------------------------------------------------
    example     = 2;

    Re          = 100;              % Reynolds number
    alpha       = 0.001;             % constant measuring the control

    N           = 80;              % spatial (inner) discretization points
    m           = 100;              % time discretization points
    nu          = 1/Re;             % Reciprocal of Reynolds number
    p           = 10;
    k           = 15;

    solverDis   = 'bEuler';
    romType     = 'DEIM';

    doPlots     = 1;
    LaTeX       = 1;
%-------------------------------------------------------------------------
%  Adjust example parameters
%-------------------------------------------------------------------------
    x0          = 0;
    x1          = 1;

    if example == 1
        T       = 10;
        y0      = @initial_cond1;       % function handle initial condition
        u0      = @initial_control1;    % function handle forcing term
    else
        T       = 1;
        y0      = @initial_cond2;       % function handle initial condition
        u0      = @initial_control2;    % function handle forcing term
    end
    domain      = [x0 x1 T];
%-------------------------------------------------------------------------
%  Initialize Trust Region parameters
%-------------------------------------------------------------------------
    delta0      = 2;
    eta1        = 0.25;
    eta2        = 0.75;
    gamma1      = 0.5;
    gamma2      = 0.5;
    gamma3      = 2;
    gamma4      = 0.95;
    l           = 0;                % iteration index

    grad_err    = 10^-8;            % stopping criterion
```

```
%-------------------------------------------------------------------------%
%  Initialize discretization
%-------------------------------------------------------------------------%
    x           = linspace(0,1,N+2);    % spatial grid
    xinner      = x(2:end-1);
    t           = linspace(0,T,m);      % time grid

    if example == 1
        Yd      = desired_state1(t,x);  % desired state
    else
        Yd      = desired_state2(t,x);  % desired state
    end
%-------------------------------------------------------------------------%
%  TRPOD algorithm
%-------------------------------------------------------------------------%
    % Set initial values
    u       = u0(t,x(2:end-1),nu);
    delta   = delta0;
    data_plot(romType,t,x,u,1,1);

    % Compute the snapshot matrix for the inital control
    [Yfem,t,timeFull,M,nonlinSNAP] = burgersFD(nu,y0,u,domain,N,m,solverDis);
    data_plot(romType,t,x,Yfem,1,0);
    fprintf('time FEM solver:\t%f\n',timeFull);

    figure();
    mesh(xinner,t,nonlinSNAP');

    % compute ROM
    [V,Ar,precon,precon2,nlin,Pin,yr0,timePOD] = generateROM(romType,nu,domain
        ,Yfem,nonlinSNAP,N,M,p,k,y0(x));
    fprintf('time compute ROM:\t%f\n',timePOD);

    % integrate ROM
    [Yrom,timeROM,~] = integrateROM(romType,domain,V,Ar,precon,precon2,nlin,
        Pin,yr0,y0(x),u,m);
    fprintf('time integrate ROM:\t%f\n',timeROM);
    data_plot(romType,t,x,Yrom,1,3);

    % Cost evaluation
    fk  = cost_evaluation(Yfem,Yd,u,alpha,domain);
    cost = fk;
    mk  = cost_evaluation(Yrom,Yd,u,alpha,domain);
    costR = mk;

    fprintf('cost full model:\t%f\n',fk);
    fprintf('cost reduced model:\t%f\n',mk);

    % compute gradient
    [grad,timeGrad] = gradientROM(romType,domain,Yd,V,Ar,precon,precon2,nlin,
        Pin,N,m,Yrom,u,alpha,mk,@norm_gradient,yr0,y0);
    fprintf('time compute gradient:\t%f\n',timeGrad);
    data_plot(romType,t,x,grad,1,2);

```

```
124     % iteration
125     while norm_gradient(grad,domain) > grad_err
126         l       = l+1;
127 
128         % minimiye the model function
129         gradN   = grad/norm_gradient(grad,domain);
130         [sk,mk_new] = line_search(romType,domain,Yd,V,Ar,precon,precon2,nlin,
                Pin,yr0,y0,m,u,alpha,delta,-gradN);
131 
132         fprintf('aktual step size:\t%f\n',sk);
133 
134         u_new = u - sk*gradN;
135         data_plot(romType,t,x,grad,l,2);
136 
137         % Step 4: compute cost functional
138         % FEM used here can be used to determine the next POD basis
139         [Y_new,~,~,M,nonlinSNAP] = burgersFD(nu,y0,u_new,domain,N,m,solverDis)
                ;
140         data_plot(romType,t,x,Y_new,l,0);
141         fk_new = cost_evaluation(Y_new,Yd,u_new,alpha,domain);
142 
143         fprintf('\n\niteration l=%d\n',l);
144         fprintf('cost full model:\t%f\t%f\n',fk_new,fk);
145         fprintf('cost reduced model:\t%f\t%f\n',mk_new,mk);
146 
147         pk = (fk-fk_new)/(mk - mk_new);
148         % Step 5: update
149         if pk >= eta1
150             % this means the iteration was successful
151             p = p+1;
152             k = k+1;
153             fk = fk_new;
154             cost = [cost fk];
155             mk = mk_new;
156             costR = [costR mk];
157             u = u_new;
158             data_plot(romType,t,x,u,l,1);
159 
160             Y = Y_new;
161             % compute ROM
162             [V,Ar,precon,precon2,nlin,Pin,yr0,timePOD] = generateROM(romType,
                    nu,domain,Yfem,nonlinSNAP,N,M,p,k,y0(x));
163             fprintf('time compute ROM:\t%f\n',timePOD);
164 
165             % compute gradient
166             [grad,timeGrad] = gradientROM(romType,domain,Yd,V,Ar,precon,
                    precon2,nlin,Pin,N,m,Yrom,u_new,alpha,mk,@norm_gradient,yr0,y0
                    );
167             fprintf('time compute gradient:\t%f\n',timeGrad);
168             data_plot(romType,t,x,grad,l,2);
169 
170             fprintf('current norm of the gradient:\t%f\n',norm_gradient(grad,
                    domain));
171 
```

```
            % update trust region radius
            if pk > eta2
               delta =gamma3*delta;     % increase trust region radius
               disp('case 1');
            else
               delta = gamma2*delta;    % decrease trust region radius
               disp('case 2');
            end
        else
            % this means the iteration was NOT successful
            % decrease trust region radius
            delta = min(gamma4*sk,gamma1*delta);
            disp('case 3');
            l = l-1;
        end
        fprintf('new trust-region radius:\t%f\n',delta);
        if delta <= 10e-2
            break;
        end
    end
end

%-------------------------------------------------------------------------
%  Supporting functions
%-------------------------------------------------------------------------
function y = L2norm(f,domain)
    % define parameters and discretization matrices
    [n,m]   = size(f);      %n = spatial, m = temporal
    h       = (domain(2)-domain(1))/(n+1);
    dT      = domain(3)/(m-1);

    W1      = speye(n,n);
    W1(1,1) = 1/2;
    W1(n,n) = 1/2;
    W1      = h*W1;

    W2      = speye(m,m);
    W2(1,1) = 1/2;
    W2(m,m) = 1/2;
    W2      = dT*W2;

    fx      = zeros(m,1);
    for i=1:m
        fx(i) = f(:,i)'*W1*f(:,i);
    end

    y = sum(W2*fx);
end

function y = initial_cond1(x)
    y = sin(pi*x)';
end

function y = initial_control1(t,x,nu)
```

```
226     y = zeros(length(x),length(t));
227     for i=1:length(x)
228         for j=1:length(t)
229             y(i,j) = (nu*pi^2-1)*exp(-t(j))*sin(pi*x(i))+ pi*exp(-2*t(j))*sin(
                    pi*x(i))*cos(pi*x(i));
230         end
231     end
232 end
233
234 function y = initial_cond2(x)
235     y= +(x<=.5)';
236 end
237
238 function y = initial_control2(t,x,nu)
239     y = zeros(length(x),length(t));
240 end
241
242 function y = desired_state1(t,x,nu)
243     y = zeros(length(x),length(t));
244     for i=1:length(x)
245         for j=1:length(t)
246             y(i,j) = sin(pi*x(i))*exp(-t(j));
247         end
248     end
249 end
250
251 function [sol] = desired_state2(t,x,nu)
252     sol = zeros(length(x),length(t));
253 end
254
255 function y = norm_gradient(g,domain)
256     [n,m] = size(g);
257
258     g2 = zeros(n+2,m);
259     g2(2:end-1,:) = g;
260     y = sqrt(L2norm(g2,domain));
261 end
262
263 function [] = data_plot(romType,t,x,Y,k,type)
264     t_initial = 0;
265     t_final = 1;
266     doPlots = 1;
267
268     if doPlots == 1
269         font_size = 15;
270
271         control_min = -10;
272         control_max = 1;
273
274         gradient_min = -2*10^-4;
275         gradient_max = 2*10^-4;
276
277         model_min = -.2;
278         model_max = 1.2;
```

```
279
280         f = figure();
281         switch type
282             % plot the control
283             case 1
284                 mesh(x(2:end-1),t(2:end),Y(:,2:end)');
285                 xlabel('x'); ylabel('t');
286                 axis([0 1 t_initial t_final control_min control_max]);
287                 file = sprintf(strcat('images/TRPOD/',romType,'-iteration-%d-
                        control'),k);
288
289             % plot the gradient
290             case 2
291                 mesh(x(2:end-1),t(2:end),Y(:,2:end)');
292                 xlabel('x'); ylabel('t');
293                 axis([0 1 t_initial t_final gradient_min gradient_max]);
294                 file = sprintf(strcat('images/TRPOD/',romType,'-iteration-%d-
                        gradient'),k);
295
296             % plot the reduced model
297             case 3
298                 mesh(x,t,Y');
299                 xlabel('x'); ylabel('t');
300                 axis([0 1 t_initial t_final model_min model_max]);
301                 file = sprintf(strcat('images/TRPOD/',romType,'-iteration-%d-
                        ROM'),k);
302
303             % plot the model
304             otherwise
305                 mesh(x,t,Y');
306                 xlabel('x'); ylabel('t');
307                 axis([0 1 t_initial t_final model_min model_max]);
308                 file = sprintf(strcat('images/TRPOD/',romType,'-iteration-%d-
                        FEM'),k);
309         end
310
311         set(gca,'FontSize',font_size);
312         h_xlabel = get(gca,'XLabel');
313         h_ylabel = get(gca,'YLabel');
314         set(h_xlabel,'FontSize',font_size);
315         set(h_ylabel,'FontSize',font_size);
316         print(f, '-depsc', file);
317     end
318 end
```